MySQL Manuel de Référence pour 3.23.5−alpha.

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

1 Informations générales à propos de *MySQL*

Ceci est la manuel de référence de ***MySQL***; il correspond à la version 3.23.2−alpha de ***MySQL***.

MySQL est un langage rapide, multi−threaded, multi−utilisateur, c'est aussi un serveur de base de données SQL robuste (Structured Query Language).

Pour les plates−formes Unix et OS/2, ***MySQL*** est libre; pour les plates−formes Microsoft il est nécessaire d'acquitter une licence après un temps d'essai de 30 jours. 3 Support et licences MySQL).

The ***MySQL*** home page fournit les dernières informations concernant ***MySQL***.

Pour une description des capacités de ***MySQL***, 1.4 Caractéristiques de MySQL.

Pour les instructions d'installation, cf. section G Commentaires sur le portage vers d'autres systèmes d'exploitation.

Pour une information sur la mise à jour de la version 3.21, 4.16.2 Mise à jour de a 3.21 vers 3.22.

Pour un exemple complet, 8 Exemple MySQL.

Pour des informations sur SQL et les benchmark, reportez vous au dossier de benchmark. Dans les distributions source, il est situé dans le dossier ``bench''. Dans les distributions compilées, il est situé dans le dossier ``sql−bench''

Pour un historique des caractéristiques et des bogues corrigés, D Historique des versions de MySQL.

Pour une liste des bogues connus et des lacunes de MySQL, E Erreurs connues et manques de ***MySQL***.

Pour les prochaines fonctionnalités, F Liste de voeux pour les versions futures de MySQL (la TODO).

Pour la liste de tous les contributeurs à ce produit, C Contributions à MySQL.

IMPORTANT:

Envoyer les bugs ou erreurs, questions ou commentaires à la liste de diffusion : mysql@lists.mysql.com. 2.3 Comment rapporter des bugs et des problèmes.

Dans les distributions source, le script `mysqlbug` est rangé dans le dossier ``scripts''. Dans les distributions compilées, `mysqlbug` est rangé dans le dossier ``bin''.

Si vous avez des suggestions concernant les ajouts de fonctionnalités ou corrections à ce manuel, envoyer les à la liste de diffusion (en anglais) (@email:{mysql@lists.mysql.com}), avec le sujet suivant : `documentation suggestion : [Insert topic here]`. 2.1 Les listes de diffusion MySQL

1.1 Qu'est ce que MySQL?

MySQL est un véritable serveur de base de données SQL multi−utilisateur et multi−threaded. SQL est le plus populaire langage de base de données dans le monde. ***MySQL*** est une configuration client/serveur ce qui

consiste en un serveur démon `mysqld`, différents programmes clients et des librairies.

SQL est un langage standardisé qui rend facile le stockage, la mise à jour et l'accès à l'information. Par exemple, vous pouvez utiliser le SQL pour récupérer des informations sur un produit ou stocker des informations client sur un site web. ***MySQL*** est suffisamment rapide et flexible pour gérer des historiques et des images.

Les principaux objectifs de ***MySQL*** sont la rapidité, la robustesse et la facilité d'utilisation. ***MySQL*** a été originellement développé parce que nous au TcX avions besoin d'un serveur SQL qui puisse gérer des grandes bases de données de manière plus rapide que ce que pouvaient offrir les distributeurs de bases de données. Nous utilisons donc ***MySQL*** depuis 1996 dans un environnement de plus de 40 bases de données contenant 10,000 tables, dont plus de 500 contiennent plus de 7 millions d'enregistrements. C'est environ 100 giga octets de données critiques.

La base sur laquelle ***MySQL*** est construite est un ensemble de routines qui ont été largement éprouvées pendant des années dans un environnement de production exigeant. Même si ***MySQL*** est encore en développement, il propose déjà un ensemble de fonctionnalités riches et extrêmement utiles.

La prononciation classique de ***MySQL*** est ``Maille Ess Ku Ell''

1.2 A propos de ce manuel

Ce manuel est actuellement disponible en Texinfo, texte plein, Info, HTML, PostScript and PDF versions. A cause de leur taille, les versions en PostScript et PDF ne sont pas inclues dans la distribution principale de MySQL, mais sont disponibles sur http://www.mysql.com.

Le document initial est le fichier Texinfo. La version HTML a été générée automatiquement avec une version modifiée de `texi2html`. Les versions plain text et Info ont été produites avec `makeinfo`. La version Postscript version a été produite en utilisant `texi2dvi` et `dvips`. La version PDF a été produite avec l'utilitaire Ghostscript `ps2pdf`.

La version américaine de ce manuel a été écrite et mise à jour par David Axmark, Michael (Monty) Widenius, Paul DuBois and Kim Aldale. Pour les autres contributeurs, cf. section C Contributions à MySQL.

1.2.1 Conventions utilisées dans ce manuel

Ce manuel utilise certaines conventions typographiques :

`constant`
: La police à empattement constant est utilisée pour les noms de commandes et les options; les déclarations en SQL; les noms de base de données, tables et colonnes; le code en C et Perl; et les variables d'environnement. Exemple: ``Pour obtenir une aide succinte, utilisez l'option `--help` de `mysqladmin`.''

``filename''
: La police à empattement constant avec des guillemets est utilisée pour les noms de fichiers et les arborescences. Exemple: ``La distribution est installée dans le dossier ``/usr/local/''.''

``c''
: La police à empattement constant avec des guillemets est aussi utilisée pour indiquer une suite de caractères. Exemple: ``Pour indiquer un caractère joker, utilisez le caractère ``%''.''

`italic`
: La police italic est utilisée pour mettre en relief certains termes, *Comme ceci*.

`boldface`
: La police Gras pour un accès à des noms privilégiés (e.g., ``Ne distribuez pas le droit de ***process*** à la légère'') et souligner des ***termes spécialement importants***.

Lorsque les commandes sont décrites et doivent être exécutées par un programme spécifique, le programme est indiquée par le prompt avec la commande. Par exemple, `shell>` indique que la commande doit être exécutée à partir du login shell, et `mysql>` indique que la commande doit être exécutée à partir client `mysql`.

```
shell62; Tapez une commande shell ici

mysql62; Tapez une commande mysql ici
```

Les commandes Shell décrites utilisent la syntaxe shell Bourne. Si vous utilisez un shell `csh`, vous devez formuler vos commandes un peu différemment.

Par exemple, la séquence de mise à jour d'une variable d'environnement et d'exécution d'une commande se présente comme suit dans la syntaxe Bourne:

```
shell62; VARNAME=value some_command
```

pour `csh`:

shell> setenv VARNAME value

```
shell62; some_command
```

Les noms de base de données, tables et colonnes sont souvent utilisés dans des commandes. Pour indiquer ces utilisations le manuel utilise la convention suivante, `nom_base_de_donnees`, `nom_table` et `nom_colonne`. Par exemple, vous pourrez voir des déclarations comme les suivantes:

```
mysql62; SELECT nom_colonne FROM nom_base_de_donnees.nom_table;
```

Cela signifie que si vous devez utiliser une déclaration similaire, vos utiliserez vos propres noms de base de données, de tables et de colonnes, peut−être comme ceci :

```
mysql62; SELECT nom_auteur FROM biblio_db.liste_auteur;
```

Les déclarations SQL peuvent être écrites en majuscules ou minuscules. Lorsque ce manuel montre une expression SQL, les majuscules sont utilisées pour les mots clefs en cours de discussion et les minuscules pour le reste de l'expression. Donc vous pourrez voir ce qui suit pour la déclaration du SELECT:

```
mysql62; SELECT count(*) FROM nom_table;
```

D'autre part, si il s'agit de la fonction `COUNT()`, l'expression sera écrite comme suit:

```
mysql62; select COUNT(*) from nom_table;
```

Dans le cas où aucun mot clef n'est à souligner, l'expression est en majuscule.

Dans la description de la syntaxe d'une expression, les crochets (``['' et ``]'') sont utilisés pour indiquer les options:

```
DROP TABLE [IF EXISTS] nom_table
```

Lorsque les éléments syntaxiques consistent en un certain nombres d'alternatives, les alternatives sont séparés par des barres verticales (``|''). Quand un membre d'un ensemble de possibilités peut être choisit, les

possibilités sont listées à l'intérieure de crochets. Quand un membre doit être choisit, les possibilités sont listées à l'intérieure de (``{'' and ``''}):

```
TRIM([[BOTH | LEADING | TRAILING] [remstr] FROM] str)

{DESCRIBE | DESC} nom_table {nom_colonne | wild}
```

1.3 Historique de MySQL

Nous avions l'intention d'utiliser `mSQL` pour se connecter à nos tables fondées sur nos routines (ISAM). Après plusieurs tests, nous sommes venus à la conclusion que `mSQL` n'était pas assez rapide, ni flexible pour nos besoins. La résultat a été le codage d'une nouvelle interface SQL pour notre base ave la plupart des interface API de `mSQL`. Ces API ont été choisis pour faciliter le portage sur des codes tiers.

La dérivation du nom ***MySQL*** n'est pas parfaitement claire. Notre arborescence et un certain nombre de nos librairies et outils commencaient par le préfixe "My" depuis des années. Même si la soeur de Monty (il y a quelques années) était surnommée My. A part les deux anecdotes le nom de ***MySQL*** est encore un mystère, pour nous aussi.

1.4 Caractéristiques de MySQL

La liste suivante décrit quelques caractéristiques importantes de ***MySQL***:

- Complètement multi−threaded en utilisant les threads du noyau. Cela signifie qu'il peut utiliser plusieurs CPU.
- C, C++, Eiffel, Java, Perl, PHP, Python et TCL APIs. 20 API MySQL.
- Fonctionne sur différentes plates−formes. 4.2 Systèmes d'exploitation supportés par MySQL.
- Plusieurs types de colonnes: entier signé/non−signé 1, 2, 3, 4et 8 bytes long, `FLOAT`, `DOUBLE`, `CHAR`, `VARCHAR`, `TEXT`, `BLOB`, `DATE`, `TIME`, `DATETIME`, `TIMESTAMP`, `YEAR`, `SET` et `ENUM` types. 7.2 Types de colonnes.
- Jointures ultra rapides, gr ce à un optimiseur de jointures à une passe.
- Les requêtes `SELECT` et `WHERE` supportent tous les opérateurs et fonctions. Exemple:

```
mysql62; SELECT CONCAT(first_name, " ", last_name) FROM nom_table
           WHERE income/dependents 62; 10000 AND age 62; 30;
```

- Les fonctions SQL sont implémentées à travers des classes de librairies extrêmement optimisées et devraient fonctionner aussi vite qu'elles peuvent! En général il n'y a plus d'allocation mémoire après une requête d'initialisation.
- Support pour les clauses des instructions SQL `GROUP BY` et `ORDER BY` . Support pour le groupe d'instructions (`COUNT()`, `COUNT(DISTINCT)`, `AVG()`, `STD()`, `SUM()`, `MAX()` et `MIN()`).
- Support pour `LEFT OUTER JOIN` avec la syntaxe ANSI SQL et ODBC.
- Le mélange des tables de différentes bases de données est supporté dans une même requête (après la version 3.22).
- Système flexible et sécurisé de droits et de mots de passe, et qui autorise une vérification faites sur l'hôte. Les mots de passe sont sécurisés depuis que la gestion des mots de passe est crypté entre le client et le serveur.
- ODBC (Open−DataBase−Connectivity) pour Windows95 (avec source). Toutes les fonctions ODBC 2.5 et d'autres. Vous pouvez, par exemple, utiliser Access pour vous connecter au serveur ***MySQL***. 16 MySQL et ODBC
- Tables rapides à B−tree avec compression des index.
- 16 index par tables sont autorisés. Chaque index consiste en 1 et 15 colonnes ou parties de colonnes. La longueur maximale d'un index est de 256 bytes (ceci peut être modifié lors de la compilation ***MySQL***). Un index devrait utiliser un préfixe `CHAR` ou `VARCHAR` .
- Enregistrements de longueur fixe ou variable.
- Les Tables de hashage en mémoire, qui sont utilisées comme des tables temporaires.
- Gestion de grandes bases de données. Nous utilisons ***MySQL*** avec des bases de 50,000,000 enregistrements.
- Toutes les colonnes ont des valeurs par défaut. Vous pouvez utiliser `INSERT` pour insérer un ensemble de colonnes; les colonnes qui n'ont de valeurs explicites sont initialisées à leur valeur par défaut.
- Utilise GNU Automake, Autoconf et `libtool` pour la portabilité.
- Ecrit en C et C++. Testé avec un nombre conséquent de compilateurs différents.
- Système efficace d'allocation mémoire thread−based.
- Pas de perte de mémoire. Testé avec un détecteur de perte de mémoire commercial (`purify`).
- Inclus `isamchk`, un utilitaire rapide pour le contrôle, l'optimisation et la réparation des tables. 13 Maintenance d'un serveur MySQL
- Support de l' ISO−8859−1 Latin1. Par exemple, les caractères scandinaves @ringaccent{a}, @"a and @"o sont autorisés dans le nom des tables et des colonnes.
- Toutes les données sont sauvegardées sous le format ISO−8859−1 Latin1. Les comparaisons de colonnes en chaînes de caractères normales

sont "insensible à la casse".
- Le tri est cohérent avec la norme ISO−8859−1 Latin1. Il est possible de le changer dans le source en ajoutant un nouveau tableau de tri. Pour voir un exemple de tri avancé, regardez le code Tchèque. ***MySQL*** supporte différentes polices qui peuvent être spécifiées à la compilation.
- Alias sur les tables et colonnes comme dans le standard SQL92.
- `DELETE`, `INSERT`, `REPLACE` et `UPDATE` retournent le nombre de lignes traitées.
- Les noms de fonction restent disponibles pour les tables et les colonnes. Par exemple, `ABS` reste un nom de colonne valide. L'unique restriction est que lors de l'appel d'une fonction, les espaces entre le nom de la fonction et les parenthèses '(' qui suivent ne sont pas autorisées. 7.30 Mots reservés par MySQL
- Tous les programmes de ***MySQL*** dispose d'une aide en ligne, accessible gr ce à l'option `--help` ou `-?`.
- Le serveur peut fournir au client les messages d'erreurs en plusieurs langues. 9.1 Quels sont les langues supportés par MySQL?.
- Les clients se connectent au serveur ***MySQL*** en utilisant les connexions TCP/IP, les sockets Unix ou les pipes nommés sous NT.
- La commande spécifique à ***MySQL*** `SHOW` peut être utilisée pour avoir des informations concernant les bases de données, les tables et les index. La commande `EXPLAIN` peut être utilisée pour déterminer comment l'optimiseur résout la requête.

1.5 Est ce que MySQL est stable?

Cette section répond aux questions "Qu'elle est la stabilité de ***MySQL***?" et, ``Puis−je faire confiance à ***MySQL*** sur ce projet?". Nous allons essayer de clarifier quelques problèmes et répondre aux questions les plus importantes qui concernent une majorité de personnes. Cette section rassemble les informations récoltées dans la liste de diffusion (qui est très active dans l'identification des bogues).

A TcX, ***MySQL*** fonctionne depuis mi−1996 sans aucun problème. Lorsque ***MySQL*** a été diffusé auprès du grand public, un certain nombre de codes n'avaient pas été testés et ils ont été vite identifiés par les nouveaux utilisateurs qui ont utilisé les requêtes d'une manière différente de la notre. Chaque nouvelle release a moins de problèmes de portabilité que la précédente (même si à chaque fois de nouvelles fonctionnalités sont identifiées), et nous espérons bientôt qu'une prochaine version sera labellisée "Stable".

Chaque release de ***MySQL*** est utilisable et il n'y a de problèmes uniquement lorsque les utilisateurs commencent à utiliser les "zones d'ombres". Naturellement, il est difficile de connaître le contenu de ces zones d'ombres; cette section a pour objectif d'identifier les zones connues. La description concerne la version 3.22.x de ***MySQL***. Tous les bogues connus sont corrigés dans la dernière version, à l'exception de la liste des bogues répertoriés dans la section des erreurs. Cf. section E Erreurs connues et manques de ***MySQL***.

MySQL est décomposé en plusieurs couches et différents modules indépendants. Les modules sont listés ci−dessous avec leur état de stabilité :

Le gestionnaire de tables ISAM −− Stable
: Il gère le stockage et la lecture de toutes les données. Dans toutes les releases de ***MySQL***, un seul bogue a été reporté. La seule façon connue d'avoir une table corrompue est de "tuer" le serveur au milieu de la mise à jour. Et même un tel scénario ne détruira probablement pas toutes les données sans qu'il soit possible de les récupérer, car toutes les données sont vidées de la mémoire, entre chaque requête. Il n'y a jamais eu de bogues liés à la destruction de données.

L'analyseur syntaxique et l'analyseur lexical −− Stable
: Il n'y a pas un seul bogue d'identifié depuis longtemps dans ce module.

Le code du client C −− Stable
: Pas de problèmes connus. Dans la version précédente 3.20, il y avait quelques limites dans la taille du buffer d'envoi/réception. Depuis de la 3.21.x, la taille du buffer est maintenant dynamique et initialisé par défaut à 24M.

Programmes clients standards −− Stable
: Cela inclus `mysql`, `mysqladmin` et `mysqlshow`, `mysqldump`, et `mysqlimport`.

Basic SQL −− Stable
: Système de fonction basic SQL, classes de chaînes de caractères et gestion dynamique de la mémoire. Pas de bogues identifiés.

Optimiseur de requête −− Gamma

Optimiseur d'intervalle −− Stable

Optimiseur de jointure −− Stable

Verrouillage −− Gamma
: Cela est très dépendant du système. Sur certains systèmes il y a de gros problèmes en utilisant le verrouillage standard de l'OS (`fcntl()`). Dans ces cas là, vous devriez faire fonctionner le démon ***MySQL*** avec l'option `--skip-locking`. Les problèmes sont connus, sur quelques systèmes Linux et sur SunOS lorsque le gestionnaire de fichiers est monté sur NFS.

Linux threads −− Gamma

Le seul problème identifié concerne l'appel de la fonction `fcntl()`, qui est résolu en utilisant l'option `−−skip−locking` de `mysqld`. Quelques personnes ont reporté des problèmes avec la version release 0.5.

Solaris 2.5+ pthreads −− Stable
Nous l'utilisons pour nos systèmes en production.

MIT−pthreads (autres systèmes) −− Gamma
Il y a pas de bogues identifiés depuis la 3.20.15 et la 3.20.16. sur quelques systèmes, il y a un disfonctionnement . Quelques opérations sont assez lentes (un arrêt de 1/20 second est effectué entre chaque requête). Bien sur, MIT−pthreads peut ralentir un peu l'ensemble, mais les commandes `SELECT` qui utilisent des index sont faites en un laps de temps tellement court, que le verrouillage et l'échange de thread n'est pas nécessaire.

Autres implémentations −− Alpha − Beta
Le portage sur d'autres systèmes restent assez nouveau et ont peut−être des bugs, probablement dans le serveur ***MySQL***, et plus certainement dans l'implémentation des threads.

LOAD DATA ..., INSERT ... SELECT −− Stable
Quelques personnes ont cru identifier des bogues dans ce module, c'était lié à une méprise. Vérifier la documentation sur le sujet!

ALTER TABLE −− Gamma
Changements mineures dans la 3.22.12.

DBD −− Gamma
Maintenant maintenu par Jochen Wiedmann

mysqlaccess −− Gamma
Ecrit et maintenu par Yves Carlier

GRANT −− Beta
Gros changement dans la ***MySQL*** 3.22.12.

MyODBC (utilise ODBC SDK 2.5) −− Beta
Il fonctionne normalement avec la plupart des programmes.

TcX fournit un support payant par email, cependant la mailing liste ***MySQL*** répond à la plupart des questions. Les bogues sont en général corrigés par des patchs; pour les bogues plus importants, il y a de nouvelles versions.

1.6 Compatibilité an 2000

MySQL lui−même n'a pas de problémes avec la compatibilité An 2000 (Y2K):

- ***MySQL*** utilise les fonctions de gestion du temps d'Unix et n'a pas de problèmes jusqu'en `2069`; toutes les années sur 2−digit sont comprises entre `1970` et `2069`, cela qui signifie que si vous stocker `01` dans une colonne année, ***MySQL*** le traite comme `2001`.
- Toutes les fonctions de date de ***MySQL*** sont stockées dans un fichier unique ``` ``sql/time.cc'' ``` et codées pour être compatible An 2000.
- Dans la version 3.22 de ***MySQL*** et les versions ultérieures, le nouveau type de colonne `Année` peut stocker des années entre `0` et `1901` jusqu'à `2155` sur 1 byte et éditer sur 2 ou 4 digits.

Vous pouvez avoir des problèmes avec des applications qui utilisent ***MySQL*** mais non compatible An 2000.

Malheureusement ces problèmes sont difficilement corrigibles lorsque ces applications sont écrites par différents programmeurs et que chacun utilise ses propres conventions pour gérer les dates.

Vous trouverez ci−après une démonstration simple qui illustre le fait que ***MySQL*** n'a aucun problème avec les dates jusqu'en 2030!

```
mysql62; DROP TABLE IF EXISTS y2k;

mysql62; CREATE TABLE y2k (date date, date_time datetime, time_stamp timestamp);

mysql62; INSERT INTO y2k VALUES ("1998−12−31","1998−12−31 23:59:59",19981231235959);

mysql62; INSERT INTO y2k VALUES ("1999−01−01","1999−01−01 00:00:00",19990101000000);

mysql62; INSERT INTO y2k VALUES ("1999−09−09","1999−09−09 23:59:59",19990909235959);

mysql62; INSERT INTO y2k VALUES ("2000−01−01","2000−01−01 00:00:00",20000101000000);
```

```
mysql62; INSERT INTO y2k VALUES ("2000−02−28","2000−02−28 00:00:00",20000228000000);

mysql62; INSERT INTO y2k VALUES ("2000−02−29","2000−02−29 00:00:00",20000229000000);

mysql62; INSERT INTO y2k VALUES ("2000−03−01","2000−03−01 00:00:00",20000301000000);

mysql62; INSERT INTO y2k VALUES ("2000−12−31","2000−12−31 23:59:59",20001231235959);

mysql62; INSERT INTO y2k VALUES ("2001−01−01","2001−01−01 00:00:00",20010101000000);

mysql62; INSERT INTO y2k VALUES ("2004−12−31","2004−12−31 23:59:59",20041231235959);

mysql62; INSERT INTO y2k VALUES ("2005−01−01","2005−01−01 00:00:00",20050101000000);

mysql62; INSERT INTO y2k VALUES ("2030−01−01","2030−01−01 00:00:00",20300101000000);

mysql62; INSERT INTO y2k VALUES ("2050−01−01","2050−01−01 00:00:00",20500101000000);

mysql62; SELECT * FROM y2k;


+------------+---------------------+----------------+
| date       | date_time           | time_stamp     |
+------------+---------------------+----------------+
| 1998−12−31 | 1998−12−31 23:59:59 | 19981231235959 |
| 1999−01−01 | 1999−01−01 00:00:00 | 19990101000000 |
| 1999−09−09 | 1999−09−09 23:59:59 | 19990909235959 |
| 2000−01−01 | 2000−01−01 00:00:00 | 20000101000000 |
| 2000−02−28 | 2000−02−28 00:00:00 | 20000228000000 |
| 2000−02−29 | 2000−02−29 00:00:00 | 20000229000000 |
| 2000−03−01 | 2000−03−01 00:00:00 | 20000301000000 |
| 2000−12−31 | 2000−12−31 23:59:59 | 20001231235959 |
| 2001−01−01 | 2001−01−01 00:00:00 | 20010101000000 |
| 2004−12−31 | 2004−12−31 23:59:59 | 20041231235959 |
| 2005−01−01 | 2005−01−01 00:00:00 | 20050101000000 |
| 2030−01−01 | 2030−01−01 00:00:00 | 20300101000000 |
| 2050−01−01 | 2050−01−01 00:00:00 | 00000000000000 |
+------------+---------------------+----------------+
13 rows in set (0.00 sec)

mysql62; DROP TABLE y2k;
```

Cela montre que le type `DATE` et `DATETIME` sont compatibles, alors que le type `TIMESTAMP`, qui est utilisé pour stocker la date courante, est limitée au `2030−01−01`. `TIMESTAMP` est compris entre `1970` et `2030` sur les machines 32−bits.

Même si ***MySQL*** est compatible An 2000, il est de votre responsabilité de fournir une entrée correcte. 7.2.6.1 Bug de l'an 2000 et données de types date pour les règles appliquées par ***MySQL*** pour la gestion des entrées ambiguës de données (Donnée contenant des valeurs sur 2−digit).

1.7 Informations sur le SQL

Ces livres ont été recommandés par plusieurs personnes sur la liste de diffusion ***MySQL*** :

`Judith S. Bowman, Sandra L. Emerson et Marcy Darnovsky`

`The Practical SQL Handbook: Using Structured Query Language`

```
Second Edition

Addison−Wesley

ISBN 0−201−62623−3

http://www.awl.com
```

Celui−ci aussi a eu quelques recommandations sur la liste de diffusion :

```
Martin Gruber

Understanding SQL

ISBN 0−89588−644−8

Publisher Sybex 510 523 8233

Alameda, CA USA
```

Un tutorial SQL est disponible sur http://www.geocities.com/ResearchTriangle/Node/9672/sqltut.html

SQL in 21 Tagen (Livre en Allemand): http://www.mut.de/leseecke/buecher/sql/inhalt.htm

1.8 Liens utiles

1.8.1 Outils de développement WEB qui supportent *MySQL*.

- http://www.php.net/
- http://xsp.lentus.se/
- http://www.voicenet.com/~zellert/tjFM
- http://www.wernhart.priv.at/php/
- http://www.dbwww.com/
- http://www.daa.com.au/~james/www−sql/
- http://www.minivend.com/minivend/
- http://www.heitml.com/
- http://www.metahtml.com/
- http://www.binevolve.com/
- http://hawkeye.net/
- http://www.fastflow.com/
- http://www.wdbi.net/
- http://www.webgroove.com/
- http://www.ihtml.com/
- ftp://ftp.igc.org/pub/myodbc/README
- http://calistra.com/MySQL/
- http://www.webmerger.com Cet outil CGI interprète des fichiers, et génère des affichages dynamiques, basé sur un ensemble simple de balises. Il est fonctionne déjà avec ***MySQL*** et PostgreSQL via ODBC.

1.8.2 Serveurs Web avec des outils *MySQL*.

- http://bourbon.netvision.net.il/mysql/mod_auth_mysql/
- http://www.roxen.com/

1.8.3 Exemples et sources

- http://www.little6.com/about/linux/ Un site d'emploi en ligne, construit ***MySQL***, PHP3 et Linux.
- http://www.delec.com/is/products/prep/examples/BookShelf/index.html Un outil qui rend très simple la mise la création de documentation automatique. Ils ont utilisé ***MySQL*** comme un exemple.
- http://shredder.elen.utah.edu/steve.html Utilise ***MySQL*** et webmerger. Il y a une base des employés, et une base des plaques

d'immatriculation. (plus de 1.2 million). Le champs des immatriculation est indexé, et les recherches sont instantanées.
- http://www.worldrecords.com Un moteur de recherche pour la musique, fait avec ***MySQL*** et PHP.
- http://webdev.berber.co.il/
- http://www.webtechniques.com/features/1998/01/note/note.shtml
- http://modems.rosenet.net/mysql/
- http://www.odbsoft.com/cook/sources.htm
- http://www.gusnet.cx/proj/telsql/.
- http://www.productivity.org/projects/mysql/
- http://www.iserver.com/support/contrib/perl5/modules.html
- http://tecfa.unige.ch/guides/java/staf2x/ex/jdbc/coffee−break
- http://www.spade.com/linux/howto/PostgreSQL−HOWTO−41.html
- http://www.ooc.com/
- http://www.pbc.ottawa.on.ca/
- http://keilor.cs.umass.edu/pluribus/ Pluribus, is a free search engine that learns to improve the quality of its results over time. Pluribus works by recording which pages a user prefers among those returned for a query. A user votes for a page by selecting it; Pluribus then uses that knowledge to improve the quality of the results when someone else submits the same (or similar) query. Uses PHP and ***MySQL***.
- http://www.stopbit.com/ A technology news site using ***MySQL*** and PHP
- http://www.jokes2000.com/scripts/
- http://www.hotwired.com/webmonkey/backend/tutorials/tutorial1.html
- http://www.geocities.com/CapeCanaveral/2064/mysql.html
- http://futurerealm.com/forum/futureforum.cgi
- http://www.linuxsupportline.com/~kalendar/ KDE based calendar manager The calendar manager has both single user (file based) and multi user (***MySQL*** database) support.
- http://tim.desert.net/~tim/imger/
- http://www.penguinservices.com/scripts

1.8.4 Liens *ODBC.*

- http://www.openlinksw.com/iodbc/
- http://users.ids.net/~bjepson/freeODBC/

1.8.5 Liens *API*.

- http://www.jppp.com Partially implemented TDataset−compatible components for ***MySQL***.
- http://www.riverstyx.net/qpopmysql/ A patch to allow POP3 authentication from a MySQL database. There's also a link to someone who wrote a patch for Procmail to allow any MTA to deliver to users in a ***MySQL*** database.
- http://www.pbc.ottawa.on.ca
- http://www.lilback.com/macsql/

1.8.6 Clients et applications.

- http://www.urbanresearch.com/software/utils/urbsql/index.html
- http://www.etu.info.unicaen.fr/~fbilhaut/kmysql/

1.8.7 Autres liens *MySQL*.

- http://www.wix.com/mysql−hosting
- http://www.open.com.au/products.html
- http://www.stonekeep.com/pts/
- http://tomato.nvgc.vt.edu/~hroberts/mot
- http://www.seawood.org/msql_bind/
- http://home.wxs.nl/cgi−bin/planeteers/pgidszoek.cgi
- http://www.cynergi.net/non−secure/exportsql/
- http://SAL.KachinaTech.COM/H/1/MYSQL.html
- http://www.infotech−nj.com/itech/index.shtml
- http://www.pmpcs.com/
- http://www.aewa.org
- http://abattoir.cc.ndsu.nodak.edu/~nem/mysql/udf/
- http://21ccs.com/~gboersm/y2kmatrix/

1.8.8 SQL et interfaces pour bases de données.

- http://www.penguinpowered.com/~kmysql KMySQL is a database client for KDE that primarily supports ***MySQL***.
- http://java.sun.com/products/jdbc/
- http://tfdec1.fys.kuleuven.ac.be/~michael/fpc−linux/mysql
- http://www.gagme.com/mysql
- http://www.amsoft.ru/easysql/
- http://www.lightlink.com/hessling/rexxsql.html
- http://www.binevolve.com/~tdarugar/tcl−sql

1.8.9 Liens sur les bases de données.

- http://www.pcslink.com/~ej/dbweb.html
- http://black.hole−in−the.net/guy/webdb/
- http://www.symbolstone.org/technology/perl/DBI/index.html
- http://www−public.rz.uni−duesseldorf.de/~tolj
- http://dbasecentral.com/
- http://www.Tek−Tips.com Tek−Tips Forums are 800+ independent peer−to−peer non−commercial support forums for Computer Professionals. Features include automatic e−mail notification of responses, a links library, and member confidentiality guaranteed.

Il y a beaucoup de sites Web qui utilisent ***MySQL***. Cf. section A Quelques utilisateurs MySQL.

2 La liste de diffusion MySQL et comment rapporter des bugs

2.1 Les listes de diffusion MySQL

Pour s'incrire sur la liste de diffusion (en anglais) de ***MySQL***, envoyez un message à l'adresse suivante : mysql−subscribe@lists.mysql.com.

Pour se désinscrire de la liste de diffusion ***MySQL***, envoyez un message à l'adresse suivante : mysql−unsubscribe@lists.mysql.com.

Seule l'adresse d'origine de votre message sera utilisée. Le sujet et le contenu du mail sont ignorés.

Si votre adresse de réponse n'est pas valide, vous pouvez la spécifier explicitement. Vous pouvez alors ajouter un signe tiret `-', suivi de votre adresse, dont l'arobase `@' a été remplacé par `='. Par exemple, pour inscrire l'adrsse `paul@hote.domaine`, envoyez un message à l'adresse `mysql−subscribe−paul=hote.domaine@lists.mysql.com`.

Les mails envoyés à mysql−subscribe@lists.mysql.com ou mysql−unsubscribe@lists.mysql.com sont gérés automatiquement par le robot de la liste, ezmlm. Des informations à propos de ezmlm sont disponibles à the ezmlm Website.

Pour envoyer un message à la liste elle même, envoyez votre message à `mysql@lists.mysql.com`. Cependant, n'envoyez *pas* de message d'inscription ou de désinscription à mysql@lists.mysql.com, car les mails envoyés la bas sont automatiquement envoyé à des milliers d'utilisateurs.

Votre site local peut disposer de sa propre base d'utilisateurs de mysql@lists.mysql.com. Dans ce cas, il a peut être sa propre liste de diffusion, qui relaient les messages de `lists.mysql.com`. Dans ce cas, contactez votre administrateur pour être ajouté ou supprimé de la liste de diffusion.

Les listes de diffusions suivantes existent :

annonces
: Cette liste sert à annoncer les nouvelles versions de ***MySQL*** et des programmes associés. Elle a un faible traffic, et il est préférable que tous les utilisateurs y soient inscrit.

mysql
: La liste principale de discussion à propos de ***MySQL***. Notez bien que certains sujets sont traités dans des listes de diffusions séparées. Si vous postez un message sur une mauvaise liste, vous risquez de ne pas avoir de réponse.

mysql−digest
: La liste `mysql` dans un format condensé. Cela signifie que tous les messages de la journée sont rassemblés en un grand message, mais une fois par jour.

java
: Discussion à propos de ***MySQL*** et Java. Généralement, à propos des pilotes JDBC.

java−digest
: La version condensée de la liste sur `java`.

win32
: Tout ce qui concerne ***MySQL*** sur les OS Microsoft, comme par exemple Windows NT.

win32−digest
: La version condensée de la liste sur `win32`.

myodbc
: Tout ce qui concerne les connexions à ***MySQL*** avec ODBC.

myodbc−digest
: La version condensée de la liste sur `myodbc`.

msql−mysql−modules
: Tout ce qui concerne les connexions à ***MySQL*** avec Perl.

msql−mysql−modules−digest
: La version condensée de la liste sur `msql−mysql−modules`.

developer
: Une liste pour ceux qui travaille sur le code de ***MySQL***.

developer−digest
: La version condensée de la liste sur `developer`.

Vous vous inscrivez ou désincrivez de ces listes de la même manière que décrit ci dessus. Dans votre message, utilisez simplement l'adresse de la liste qui vous interesse, plutôt que `mysql`. Par exemple, pour s'inscrire ou se désinscrire à la liste `myodbc` envoyez un message à myodbc−subscribe@lists.mysql.com ou myodbc−unsubscribe@lists.mysql.com.

2.2 Poser des questions et rapporter des bugs

Avant d'envoyer un rapport de bug, ou une question, suivez les instructions suivantes :

- Commencez par chercher dans le manuel en ligne de ***MySQL*** :
  `http://www.mysql.com/Manual_chapter/manual_toc.html`

  Nous essayons de garder le manuel à jour en le modifiant suivant, et en ajoutant des solutions pour les problèmes rencontrés.

- Recherchez dans les archives de la liste ***MySQL***:
  `http://www.mysql.com/doc.html`

- Vous pouvez aussi utiliser http://www.mysql.com/search.html pour rechercher dans les pages web (y compris le manuel) qui sont situés à http://www.mysql.com/.

Si vous ne trouvez pas d'information dans le manuel, ou dans les pages d'archives, vérifiez avec votre expert ***MySQL*** local. Si vous ne pouvez trouver aucune réponse, passez à la section suivante pour savoir comment envoyer un message à mysql@lists.mysql.com.

2.3 Comment rapporter des bugs et des problèmes

Ecrire un rapport de bug nécessite de la patience, mais ce premier geste va économiser votre temps, et le notre. Cette section vous aide à écrire votre rapport de bug correctement, de manière à ce que vous ne perdiez pas votre temps à écrire des messages qui ne nous servirons à rien.

Nous vous encourageons à utiliser le script `mysqlbug` pour générer un rapport de bug (ou un rapport sur n'importe quel problème). `mysqlbug` est situés dans le dossier de la distribution source, ou, dans la distribution binaire, dans le dossier `` `bin' `` de votre dossier d'installation ***MySQL***.Si vous ne pouvez pas utiliser `mysqlbug`, il est préférable d'inclure toutes les informations listées ci dessous.

Le script `mysqlbug` vous aide à générer un rapport de bug en déterminant les informations suivantes automatiquement : mais si vous pensez que quelque chose d'important manque, ajoutez le au message. Relisez attentivement cette section, et assurez vous que toutes les informations décrites ci dessous sont incluses dans votre rapport.

Gardez à l'esprit qu'il est toujours possible de répondre à un message qui contient trop d'information, alors que ce n'est pas possible avec un message qui en contient pas assez. Souvent, les rapports omettent des informations car les utilisateurs pensent avoir compris les causes du problème et que certains détails sont insignifiants. Le bon principe est le suivant : Si vous doutez de quelque chose, dites le. Il est milles fois plus

rapide d'ajouter quelques lignes dans votre rapport, plutôt que d'être forcé de le renvoyé encore une fois, pour complément d'information.

L'erreur la plus répandue est que les utilisateurs n'indique pas le numéro de version de la distribution ***MySQL*** qu'ils utilisent, ou la plate forme sur laquelle ils opèrent (y compris la version de cette plate forme). C'est une information primordiale, et 99% des cas sont inexploitables sans cette information. Souvent, nous avons des questions du genre : ``Pourquoi est ce que ça plante chez moi?" et nous nous aperçevons que cette fonctionnalité n'est pas disponible sur la version de ***MySQL*** utilisée, ou bien que le bug a été corrigé dans les versions plus récentes. Parfois, l'erreur dépend de l'OS. Dans ces cas, il est presque impossible de corriger l'erreur sans connaitre le nom de l'OS, et le numéro de version de la plate forme.

N'oubliez pas d'inclure des inforamtions sur les compilateurs, si cela a un rapport avec votre problème. Souvent, on trouve des erreurs dans les compilateurs, et les utilisateurs pensent que c'est lié à ***MySQL***. La plus part des compilateurs sont en développement constants, et s'améliorent de version en version. Pour savoir si votre problème dépend du compilateur, nous avons besoin de savoir quel compilateur est utilisé. Notez que chaque problème lié à la compilation doit être considéré comme un bug, et rapporté de manière adéquate.

Une bonne description du problème est toujours utile dans un rapport de bug. C'est à dire, toutes les manipulations que vous avez faites, qui ont conduit au bug, et la description du bug lui même. Les meilleurs rapports inclus aussi un exemple complet pour reproduire le bug.

Si un programme produit un message d'erreur, il est très important d'inclure le message dans votre rapport! Si vous recherchez dans les archives, il est préférable d'utiliser le message d'erreur exact (même la casse est importante). N'essayez pas de vous souvenir du message : faites en un copier/coller du message entier!

Pensez à inclure les informations suivantes dans votre rapport:

- Le numéro de version de la distribution de ***MySQL*** que vous utilisez (par exemple, ***MySQL*** 3.22.22). Vous pouvez savoir le numéro de version que vous utilisez en exécutant la commande `mysqladmin version`. `mysqladmin` est situé dans le dossier `` `bin' `` de votre dossier d'installation ***MySQL***.
- La marque et le modèle de machine que vous exploitez.
- Le nom de l'OS et sa verison. Pour la plus part des OS, vous pouvez avoir ces informations avec la commande UNIX `uname −a`.
- Parfois, la quantité de mémoire (réelle et virtuelle) est importante. En cas de doute, incluez ces valeurs.
- Si vous utilisez une version source de ***MySQL***, le nom et la version du compilateur est aussi apréciable. Si vous avez une version binaire, le nom de la distribution est nécessaire.
- Si le problème intervient lors de la compilation, incluez l'erreur exacte, le message d'erreur, et quelques lignes de contexte.
- Si une table est liée au problème, incluez le résultat de la commande `mysqldump −−no−data nom_base_de_donnees nom_table1 nom_table2 ...` C'est très simple à faire, et c'est une méthode puissante pour rassembler toutes les informations d'une table qui nous permettra de recréer une situation identique à la votre.
- Pour les bugs liés à la vitesse, ou les problèmes avec les commandes `SELECT` vous devriez toujours include le résultat d'une commande `EXPLAIN SELECT ...`, et au moins le nombre de lignes que votre commande `SELECT` doit produire. Plus vous pourrez transmettre d'informations, sur votre situation, plus nous pourrons vous aider. Par exemple, voici un très bon rapport de bug (posté avec le script `mysqlbug`, bien entendu) : Exemple lancé depuis la ligne de commande `mysql`:

```
mysql62; SHOW VARIABLES;
mysql62; EXPLAIN SELECT ...
       60;output−from−EXPLAIN62;
mysql62; FLUSH STATUS;
mysql62; SELECT ...
       60;Une courte description du résultat de SELECT,
       en incluant le temps de travail de la requête62;
mysql62; SHOW STATUS;
       60;output from SHOW STATUS62;
```

- Si un bug ou un problème intervient lors du fonctionnement de ***MySQL***, essayer de nous fournir un script qui pourra reproduire l'anomalie. Ce script incluera tous les fichiers nécessaires. Plus le script sera proche de votre environnement, et mieux ce sera. Si vous ne pouvez pas fournir de script, vous devez au moins nous fournir l'affichage de `mysqladmin variables extended−status processlist` dans votre mail, pour nous fournir des informations sur votre système.

- Si vous pensez que ***MySQL*** produit un résultat un peu inattendu, pensez à inclure non seulement l'étrange résultat de votre requête, mais aussi votre avis sur ce qui devrait être retourné, et ce qui le justifie.
- Lorsque vous donnez un exemple de problème, il est mieux d'utiliser des noms de variables, de tables, qui existent dans votre situation réelle, plutôt que d'inventer de nouveau noms. Le problème peut être lié aux noms de colonnes, table...etc! Ces cas sont peut être rares, mais il vaut mieux être prudent que navré. Après tout, il est plus facile de fournir un exemple proche de votre exemple et c'est aussi plus clair pour nous. Dans le cas où vous ne souhaitez pas afficher vos données à des tiers, vous pouvez utiliser le dossier `ftp` pour le transférer : ftp://www.mysql.com/pub/mysql/secret/. Si les données sont vraiment très confidentielles, et que vous ne voulez pas nous les montrer, utilisez d'autres valeurs, mais considérez ce choix en dernier recours!
- Incluez toutes les options utilisées par les programmes adéquats, si possible. Par exemple, vous pouvez indiquer les options de démarrage de `mysqld`, et celle que vous utilisez pour les clients ***MySQL***. Les options de programmes tels que `mysqld` et `mysql`, et le script `configure` sont souvent la clé de mystères. Ce n'est jamais une mauvaise idée de nous les transmettre, de toutes manières. Si vous utilisez des modules, tels que Perl ou PHP, n'oubliez pas de donner leur numéro de version.
- Si vous n'arrivez pas à reproduire l'erreur avec quelques lignes, ou que la table est trop grosse pour être postée sur la liste de diffusion (plus de 10 lignes), il est préférable de faire un dump de la table, avec `mysqldump` et de créer un fichier `` `README' `` qui décrit votre problème. Créez une archive compressée de vos fichiers avec `tar` et `gzip` ou `zip`, et transférez en `ftp` dans le dossier ftp://www.mysql.com/pub/mysql/secret/. Puis, envoyez une courte description de votre problème sur la liste de diffusion.
- Si vos questions sont liées aux droits, ajoutez les données de `mysqlaccess`, celui de `mysqladmin reload` et tous les messages d'erreur que vous obtenez en vous connectant. Lorsque vous testez les droits, il est préférable de faire tourner d'abord `mysqlaccess`. Après ça, excécutez la commande `mysqladmin reload version`, et en dernier ressort, essayez de vous connectez avec le programme qui pose problème. `mysqlaccess` est situé dans le dossier `` `bin' `` de votre dossier d'installation.
- Si vosu avez un patch pour un bug, c'est bien. Mais ne supposez jamais que ce patch est tout ce dont nous révions, ou même que nous allons l'utiliser si vous ne fournissez pas les informations nécessaires pour tester le bug. Nous pouvons rencontrer des problèmes avec votre patch, voire même, nous pouvons ne pas le comprendre. Dans ce cas, nous ne pourrons pas l'utiliser. Si nous ne pouvons pas vérifier l'objectif du patch, nous ne l'utiliserons pas. Les cas de tests nous seront d'un précieux secours. Montrez bien toutes les situations que le patch prend en compte. Si nous trouvons un effet de bord (même rare), ou un cas extrême que le patche ne gère pas, il risque d'être inutile.
- Toutes les tentatives de deviner l'origine du bug, pourquoi il survient, ou de quoi il découle sont généralement fausses. Il arrive que même nous, avec l'aide d'un débugger, ne pouvons pas toujours determiner la cause réelle du bug.
- Indiquez dans votre mail que vous avez vérifié le manuel de référence, et les archives, de manière à montrer que vous avez déjà essayé de résoudre le problème vous même.
- Si vous avez une erreur `parse error`, prenez le temps de bien vérifier votre syntaxe. Si vous ne pouvez pas trouver d'erreur, il est fort probable que votre version de ***MySQL*** ne supporte par la requête que vous utilisez. Si vous utilisez la version courant et que le manuel de référence http://www.mysql.com/doc.html ne couvre pas la syntaxe que vous utilisez, c'est que ***MySQL*** ne supporte pas votre requête. Dans ce cas, votre seul option est d'implémenter vous même l'option ou d'envoyer un email à mysql−licensing@mysql.com et demandez à ce qu'elle soit implémentée. Si le manuel couvre votre syntaxe, mais que vous avez une vieille version de ***MySQL***, vérifiez l'historique de ***MySQL*** pour voir quand la syntaxe a été implémentée. D Historique des versions de MySQL. Dans ce cas, il vous reste la possibilité de vous mettre à jour avec une nouvelle version de ***MySQL***.
- Si vous avez un problème tel que vos données semblent corrompues, ou que vous avez des erreurs lorsque vous accédez à une table particulière, vous devriez essayer de vérifier et réparer vos tables avec `isamchk`. 13 Maintenance d'un serveur MySQL.
- Si vos tables se corrompent souvent, vous pouvez essayer de voir quand cela arrive, et pourquoi! Dans ce cas, `` `mysql−data−directory/'hostname'.err' `` peut contenir certaines informations sur ce qui est passé. N'oubliez pas d'ajouter toutes les informations utiles dans votre rapport de bug. Normalement, `mysqld` ne crashe ***JAMAIS*** de table s'il n'a pas été tué durant une modification. Si vous pouvez trouver la source de la fin du processus `mysqld`, il est plus facile pour nous de vous fournir un correctif.
- Si possible, téléchargez la version la plus récente de ***MySQL*** et vérifiez si cela résoud votre problème. Toutes les versions de ***MySQL*** sont testés de manière exhaustive, et devrait fonctionner sans problème. Nous essayons de rendre nos developpement compatibles avec les anciennes versions, ce qui vous permettra de changer de version de ***MySQL*** en quelques minutes! 4.3 Quelle version de MySQL utiliser.

Si vous être un client du support, n'envoyez pas le rapport en double sur l'adresse pour voir si quelqu'un expérimenté peut résoudre votre problème.

Pour savoir comment rapporter à propos de ***MyODBC***, allez à 16.2 Comment rapporter des bugs avec MyODBC.

Pour des solutions aux problèmes courants, reportez vous à 18 Problèmes et erreurs fréquents.

Lorsque des informations vous sont envoyées individuellement, et pas à la liste de diffusion, il est de bon ton de rassembler toutes les réponses, et de les envoyer à la liste de diffusion pour que le bénéfices des réponses profite à tous.

2.4 Règles pour répondre aux questions sur la liste de diffusion

Si vous pensez que vous avec une réponse d'interêt général, vous voudrez surement envoyer un mail sur la liste de diffusion, plutôt que de répondre uniquement à la personne qui a posé la question. Essayez de rendre votre réponse aussi générale que possible, pour que tous ceux qui ne connaissent pas le problème initiale puisse comprendre. Lorsque vous envoyez votre réponse, vérifiez que vous n'etes pas en double avec une autre réponse.

Dans votre réponse, essayez de résumer la question; ne vous sentez pas obligé de citer toute la question.

N'envoyez pas de mail avec votre navigateur HTML. De nombreux utilisateurs ne lisent pas leur mail avec un navigateur.

3 Support et licences MySQL

This chapter describes ***MySQL*** licensing and support arrangements, including:

- Our licensing policies for non−Microsoft and Microsoft operating systems
- The copyrights under which ***MySQL*** is distributed (3.2 Copyrights de MySQL)
- Sample situations illustrating when a license is required (3.4 Exemple de licences)
- Licensing and support costs (3.5 Licences et couts du support MySQL), and support benefits (3.6 Types de support commercial)

3.1 Politique de licence MySQL

The formal terms of the license for non−Microsoft operating systems such as Unix or OS/2 are specified in J Licence MySQL pour les sytèmes d'exploitation non−Microsoft. Basically, our licensing policy is as follows:

- For normal internal use, ***MySQL*** generally costs nothing. You do not have to pay us if you do not want to.
- A license is required if:
  - You sell the ***MySQL*** server directly or as a part of another product or service
  - You charge for installing and maintaining a ***MySQL*** server at some client site
  - You include ***MySQL*** in a distribution that is non redistributable and you charge for some part of that distribution
- For circumstances under which a ***MySQL*** license is required, you need a license per machine that runs the `mysqld` server. However, a multiple−CPU machine counts as a single machine, and there is no restriction on the number of ***MySQL*** servers that run on one machine, or on the number of clients concurrently connected to a server running on that machine!
- You do not need a license to include client code in commercial programs. The client access part of ***MySQL*** is in the public domain. The `mysql` command line client includes code from the `readline` library that is under the GNU Public License.
- For customers who have purchased 10 licenses or a high enough level of support, we provide additional functionality. Currently, this means we provide the `pack_isam` utility for creating fast compressed read−only databases. (The server includes support for reading such databases but not the packing tool used to create them.) If support agreements generate sufficient revenue, we will probably release this tool under the same license as the ***MySQL*** server.
- If your use of ***MySQL*** does not require a license, but you like ***MySQL*** and want to encourage further development, you are certainly welcome to purchase a license anyway.
- If you use ***MySQL*** in a commercial context such that you profit by its use, we ask that you further the development of ***MySQL*** by purchasing some level of support. We feel that if ***MySQL*** helps your business, it is reasonable to ask that you help ***MySQL***. (Otherwise, if you ask us support questions, you are not only using for free something into which we've put a lot a work, you're asking us to provide free support, too.)

For use under Microsoft operating systems (Win95/Win98/WinNT), you need a ***MySQL*** license after a trial period of 30 days, with the exception that licenses may be obtained upon request at no cost for educational use or for university− or government−sponsored research settings. K Licence MySQL pour les OS Microsoft. A shareware version of ***MySQL***−Win32 that you can try before buying is available at http://www.mysql.com/mysql_w32.htmy. After you have paid, you will get a password that will enable you to access the newest ***MySQL***−Win32 version.

If you have any questions as to whether or not a license is required for your particular use of ***MySQL***, please contact us. 3.5.2 Contacts.

If you require a ***MySQL*** license, the easiest way to pay for it is to use the license form at TcX's secure server at https://www.mysql.com/license.htmy. Other forms of payment are discussed in 3.5.1 Informations pratique de paiement.

3.2 Copyrights de MySQL

There are several different copyrights on the ***MySQL*** distribution:

1. The ***MySQL***−specific source needed to build the `mysqlclient` library and programs in the `` `client' `` directory is in the public domain.

Each file that is in the public domain has a header which clearly states so. This includes everything in the `client' directory and some parts of the mysys, mystring and dbug libraries.
2. Some small parts of the source (GNU getopt) are covered by the ``GNU LIBRARY LIBRARY GENERAL PUBLIC LICENSE". See the `mysys/COPYING.LIB' file.
3. Some small parts of the source (GNU readline) are covered by the ``GNU GENERAL PUBLIC LICENSE". See the `readline/COPYING' file.
4. Some parts of the source (the regexp library) are covered by a Berkeley style copyright.
5. The other source needed for the ***MySQL*** server on non−Microsoft platforms is covered by the ``MySQL FREE PUBLIC LICENSE", which is based on the ``Aladdin FREE PUBLIC LICENSE." J Licence MySQL pour les sytèmes d'exploitation non−Microsoft. When running ***MySQL*** on any Microsoft operating system, other licensing applies.

The following points set forth the philosophy behind our copyright policy:

- The SQL client library should be totally free so that it can be included in commercial products without limitations.
- People who want free access to the software into which we have put a lot of work can have it, so long as they do not try to make money directly by distributing it for profit.
- People who want the right to keep their own software proprietary, but also want the value from our work, can pay for the privilege.
- That means normal in−house use is FREE. But if you use ***MySQL*** for something important to you, you may want to help further its development by purchasing a license or a support contract. 3.6 Types de support commercial.

3.2.1 Modifications ultérieures possibles du copyright

We may choose to distribute older versions of ***MySQL*** with the GPL in the future. However, these versions will be identified as ***GNU MySQL***. Also, all copyright notices in the relevant files will be changed to the GPL.

3.3 Distribution commerciale de MySQL

This section is a clarification of the license terms that are set forth in the ``MySQL FREE PUBLIC LICENSE" (FPL). J Licence MySQL pour les sytèmes d'exploitation non−Microsoft.

MySQL may be ***used*** freely, including by commercial entities for evaluation or unsupported internal use. However, ***distribution*** for commercial purposes of ***MySQL***, or anything containing or derived from ***MySQL*** in whole or in part, requires a written commercial license from TcX AB, the sole entity authorized to grant such licenses.

You may not include ***MySQL*** ``free" in a package containing anything for which a charge is being made, except as noted below.

The intent of the exception provided in the second clause of the license is to allow commercial organizations operating an FTP server or a bulletin board to distribute ***MySQL*** freely from it, provided that:

1. The organization complies with the other provisions of the FPL, which include among other things a requirement to distribute the full source code of ***MySQL*** and of any derived work, and to distribute the FPL itself along with ***MySQL***;
2. The only charge for downloading ***MySQL*** is a charge based on the distribution service and not one based on the content of the information being retrieved (i.e., the charge would be the same for retrieving a random collection of bits of the same size);
3. The server or BBS is accessible to the general public, i.e., the phone number or IP address is not kept secret, and anyone may obtain access to the information (possibly by paying a subscription or access fee that is not dependent on or related to purchasing anything else).

If you want to distribute software in a commercial context that incorporates ***MySQL*** and you do ***not*** want to meet these conditions, you should contact TcX AB to find out about commercial licensing, which involves a payment. The only ways you legally can distribute ***MySQL*** or anything containing ***MySQL*** are by distributing ***MySQL*** under the requirements of the FPL, or by getting a commercial license from TcX AB.

3.4 Exemple de licences

This section describes some situations illustrating whether or not you must license the ***MySQL*** server. Generally these examples involve providing ***MySQL*** as part of a product or service that you are selling to a customer, or requiring that ***MySQL*** be used in conjunction with your product. In such cases, it is your responsibility to obtain a license for the customer if one is necessary. (This requirement is waived if your customer already has a ***MySQL*** license. But the seller must send customer information and the license number to TcX, and the license must be a full license, not an OEM license.)

Note that a single ***MySQL*** license covers any number of CPUs/users/customers/`mysqld` servers on a machine!

3.4.1 Vente de produits qui utilisent *MySQL*

To determine whether or not you need a ***MySQL*** license when selling your application, you should ask whether the proper functioning of your application is contingent on the use of ***MySQL*** and whether you include ***MySQL*** with your product. There are several cases to consider:

- Does your application require ***MySQL*** to function properly? If your product requires ***MySQL***, you need a license for any machine that runs the `mysqld` server. For example, if you've designed your application around ***MySQL***, then you've really made a commercial product that requires the engine, so you need a license. If your application does not require ***MySQL***, you need not obtain a license. For example, if ***MySQL*** just added some new optional features to your product (such as adding logging to a database if ***MySQL*** is used rather than logging to a text file), it should fall within normal use, and a license would not be required. In other words, you need a license if you sell a product designed specifically for use with ***MySQL*** or that requires the ***MySQL*** server to function at all. This is true whether or not you provide ***MySQL*** for your client as part of your product distribution. It also depends on what you're doing for the client. Do you plan to provide your client with detailed instructions on installing ***MySQL*** with your software? Then your product may be contingent on the use of ***MySQL***; if so, you need to buy a license. If you are simply tying into a database that you expect already to have been installed by the time your software is purchased, then you probably don't need a license.
- Do you include ***MySQL*** in a distribution and charge for that distribution? If you include ***MySQL*** with a distribution that you sell to customers, you will need a license for any machine that runs the `mysqld` server, because in this case you are selling a system that includes ***MySQL***. This is true whether the use of ***MySQL*** with your product is required or optional.
- Do you neither require for your product nor include ***MySQL*** with it? Suppose you want to sell a product that is designed generally to use ``some database'' and that can be configured to use any of several supported alternative database systems (***MySQL***, PostgreSQL, or something else). That is, your product does not not require ***MySQL***, but can support any database with a base level of functionality, and you don't rely on anything that only ***MySQL*** supports. Does one of you owe us money if your customer actually does choose to use ***MySQL***? In this case, if you don't provide, obtain or set up ***MySQL*** for the customer should the customer decide to use it, neither of you need a license. If you do perform that service, see ***MySQL*** services.

3.4.2 Vente de services *MySQL*

If you perform ***MySQL*** installation on a client's machine and any money changes hands for the service (directly or indirectly), then you must buy a ***MySQL*** license.

If you sell an application for which ***MySQL*** is not strictly required but can be used, a license may be indicated, depending on how ***MySQL*** is set up. Suppose your product neither requires ***MySQL*** nor includes it in your product distribution, but can be configured to use ***MySQL*** for those customers who so desire. (This would be the case, for example, if your product can use any of a number of database engines.)

If the customer obtains and installs ***MySQL***, no license is needed. If you perform that service for your customer, then a license is needed because then you are selling a service that includes ***MySQL***.

3.4.3 ISP *MySQL*

Internet Service Providers (ISPs) often host ***MySQL*** servers for their customers.

If you are an ISP that allows customers to install and administer ***MySQL*** for themselves on your machine with no assistance from you, neither you nor your customer need a ***MySQL*** license.

If you charge for ***MySQL*** installation and administrative support as part of your customer service, then you need a license because you are selling a service that includes ***MySQL***.

3.4.4 Utiliser un serveur web avec *MySQL*

If you use ***MySQL*** in conjunction with a web server, you don't have to pay for a license.

This is true even if you run a commercial web server that uses ***MySQL***, since you are not selling ***MySQL*** itself. However, in this case we would like you to purchase ***MySQL*** support, because ***MySQL*** is helping your enterprise.

3.5 Licences et couts du support MySQL

Our current license prices are shown below. All prices are in US Dollars. If you pay by credit card, the currency is EURO (European Union Euro) so the prices will differ slightly.

Number of licenses	*Price per copy*	*Total*
1	US $200	US $200
10 pack	US $150	US $1500
50 pack	US $120	US $6000

For high volume (OEM) purchases, the following prices apply:

Number of licenses	*Price per copy*	*Minimum at one time*	*Minimum payment*
100−999	US $40	100	US $4000
1000−2499	US $25	200	US $5000
2500−4999	US $20	400	US $8000

For OEM purchases, you must act as the middle−man for eventual problems or extension requests from your users. We also require that OEM customers have at least an extended email support contract.

If you have a low−margin high−volume product, you can always talk to us about other terms (for example, a percent of the sale price). If you do, please be informative about your product, pricing, market and any other information that may be relevant.

After buying 10 ***MySQL*** licenses, you will get a personal copy of the `pack_isam` utility. You are not allowed to redistribute this utility but you can distribute tables packed with it.

A full−price license is not a support agreement and includes very minimal support. This means that we try to answer any relevant question. If the answer is in the documentation, we will direct you to the appropriate section. If you have not purchased a license or support, we probably will not answer at all.

If you discover what we consider a real bug, we are likely to fix it in any case. But if you pay for support we will notify you about the fix status instead of just fixing it in a later release.

More comprehensive support is sold separately. Descriptions of what each level of support includes are given in 3.6 Types de support commercial. Costs for the various types of commercial support are shown below. Support level prices are in EURO (European Union Euro). One EURO is about 1.17 USD.

Type of support	*Cost per year*
Basic email support	EURO 170
Extended email support	EURO 1000
Login support	EURO 2000
Extended login support	EURO 5000

You may upgrade from any lower level of support to a higher level of support for the difference between the prices of the two support levels.

3.5.1 Informations pratique de paiement

Currently we can take SWIFT payments, cheques or credit cards.

Payment should be made to:

```
Postgirot Bank AB
105 06 STOCKHOLM, SWEDEN

TCX DataKonsult AB
BOX 6434
11382 STOCKHOLM, SWEDEN

SWIFT address: PGSI SESS
Account number: 96 77 06 − 3
```

Specify: license and/or support and your name and email address.

In Europe and Japan you can use EuroGiro (that should be less expensive) to the same account.

If you want to pay by cheque, make it payable to ``Monty Program KB" and mail it to the address below:

```
TCX DataKonsult AB
BOX 6434
11382 STOCKHOLM, SWEDEN
```

If you want to pay by credit card over the Internet, you can use TcX's secure license form.

You can also print a copy of the license form, fill it in and send it by fax to:

+46−8−729 69 05

If you want us to bill you, you can use the license form and write ``bill us" in the comment field. You can also mail a message to with your company information and ask us to bill you.

3.5.2 Contacts

For commercial licensing, or if you have any questions about any of the information in this section, please contact the ***MySQL*** licensing team. The much preferred méthode is by E−Mail to these may take much longer (Fax +46−8−729 69 05).

```
David Axmark
Detron HB
Kungsgatan 65 B
753 21 UPPSALA
SWEDEN
Voice Phone +46−18−10 22 80     (Timezone GMT+1. Swedish and English spoken)
```

3.6 Types de support commercial

3.6.1 Support email basique

Basic email support is a very inexpensive support option and should be thought of more as a way to support our development of ***MySQL*** than as a real support option.

At this support level, the ***MySQL*** mailing lists are the preferred means of communication. Questions normally should be mailed to the primary mailing list (mysql@lists.mysql.com) or one of the other regular lists (for example, mysql−win32@lists.mysql.com for Windows−related ***MySQL*** questions), as someone else already may have experienced and solved the problem you have. 2.2 Poser des questions et rapporter des bugs.

However, by purchasing basic email support, you also have access to the support address mysql−support@mysql.com, which is not available as part of the minimal support that you get by purchasing a ***MySQL*** license. This means that for especially critical questions, you can cross−post your message to mysql−support@mysql.com. (If the message contains sensitive data, you should post only to mysql−support@mysql.com.)

REMEMBER! to ALWAYS include your registration number and expiration date when you send a message to

Basic email support includes the following types of service:

- If your question is already answered in the manual, we will inform you of the correct section in which you can find the answer. If the answer is not in the manual, we will point you in the right direction to solve your problem.
- We guarantee a timely answer for your email messages. We can't guarantee that we can solve any problem, but at least you will receive an answer if we can contact you by email.
- We will help with unexpected problems when you install ***MySQL*** from a binary distribution on supported platforms. This level of support does not cover installing ***MySQL*** from a source distribution. ``Supported" platforms are those for which ***MySQL*** is known to work. 4.2 Systèmes d'exploitation supportés par MySQL.
- We will help you with bugs and missing features. Any bugs that are found are fixed for the next ***MySQL*** release. If the bug is critical for you, we will mail you a patch for it as soon the bug is fixed. Critical bugs always have the highest priority for us, to ensure that they are fixed as soon as possible.
- Your suggestions for the further development of ***MySQL*** will be taken into consideration. By taking email support you have already helped the further development of ***MySQL***. If you want to have more input, upgrade to a higher level of support.
- If you want us to help optimize your system, you must upgrade to a higher level of support.

3.6.2 Support email étendu

Extended email support includes everything in basic email support with these additions:

- Your email will be dealt with before mail from basic email support users and non−registered users.
- Your suggestions for the further development of ***MySQL*** will receive strong consideration. Simple extensions that suit the basic goals of ***MySQL*** are implemented in a matter of days. By taking extended email support you have already helped the further development of ***MySQL***.
- We include a binary version of the `pack_isam` packing tool for creating fast compressed read−only databases (it does not support `BLOB` or `TEXT` types yet). The current server includes support for reading such databases but not the packing tool used to create them.
- Typical questions that are covered by extended email support are:
  - We will answer and (within reason) solve questions that relate to possible bugs in ***MySQL***. As soon as the bug is found and corrected, we will mail a patch for it.
  - We will help with unexpected problems when you install ***MySQL*** from a source or binary distribution on supported platforms.
  - We will answer questions about missing features and offer hints how to work around them.
  - We will provide hints on optimizing `mysqld` for your situation.
- You are allowed to influence the priority of items on the ***MySQL*** TODO. This will ensure that the features you really need will be implemented sooner than they might be otherwise.

3.6.3 Support de login

Login support includes everything in extended email support with these additions:

- Your email will be dealt with even before mail from extended email support users.
- Your suggestions for the further development of ***MySQL*** will be taken into very high consideration. Realistic extensions that can be implemented in a couple of hours and that suit the basic goals of ***MySQL*** will be implemented as soon as possible.
- If you have a very specific problem, we can try to log in on your system to solve the problem ``in place."
- Like any database vendor, we can't guarantee that we can rescue any data from crashed tables, but if the worst happens we will help you rescue as much as possible. ***MySQL*** has proven itself very reliable, but anything is possible due to circumstances beyond our control (for example, if your system crashes or someone kills the server by executing a `kill −9` command).
- We will provide hints on optimizing your system and your queries.
- You are allowed to call a ***MySQL*** developer (in moderation) and discuss your ***MySQL***−related problems.

3.6.4 Support de login étendu

Extended login support includes everything in login support with these additions:

- Your email has the highest possible priority.
- We will actively examine your system and help you optimize it and your queries. We may also optimize and/or extend ***MySQL*** to better suit your needs.
- You may also request special extensions just for you. For example:

```
mysql62; select MY_CALCULATION(nom_colonne1,nom_colonne2) from nom_table;
```

- We will provide a binary distribution of all important ***MySQL*** releases for your system, as long as we can get an account on a similar system. In the worst case, we may require access to your system to be able to create a binary distribution.
- If you can provide accommodations and pay for traveler fares, you can even get a ***MySQL*** developer to visit you and offer you help with your troubles. Extended login support entitles you to one personal encounter per year, but we are as always very flexible towards our customers!

4 Instalation de MySQL

This chapter describes how to obtain and install ***MySQL***:

- For a list of sites from which you can obtain ***MySQL***, see Getting ***MySQL***.
- To see which platforms are supported, see 4.2 Systèmes d'exploitation supportés par MySQL.
- Several versions of ***MySQL*** are available, in both binary and source distributions. To determine which version and type of distribution you should use, see 4.4 Comment et quand sont générées les mises à jour.
- Installation instructions for binary and source distributions are described in 4.6 Installer une version binaire de MySQL, and 4.7 Installer une version source de MySQL. Each set of instructions includes a section on system−specific problems you may run into.
- For post−installation procedures, see 4.15 Paramètrage post−installation et tests. These procedures apply whether you install ***MySQL*** using a binary or source distribution.

4.1 Comment obtenir MySQL

Check the ***MySQL*** home page for information about the current version and for downloading instructions.

However, the Internet connection at TcX is not so fast; we would *prefer* that you do the actual downloading from one of the mirror sites listed below.

Please report bad or out of date mirrors to webmaster@mysql.com.

Europe:

- Austria [Univ. of Technology/Vienna] WWW FTP
- Bulgaria [Naturella] FTP
- Croatia [HULK] WWW FTP
- Czech Republic [Masaryk University in Brno] WWW FTP
- Denmark [Ake] WWW
- Denmark [SunSITE] WWW FTP
- Estonia [OKinteractive] WWW
- France [minet] WWW
- Finland [EUnet] WWW
- Finland [clinet] FTP
- Germany [Bonn University, Bonn] WWW FTP
- Germany [Wolfenbuettel] WWW FTP
- Germany [Staufen] WWW
- Germany [Cable & Wireless] FTP
- Greece [NTUA, Athens] WWW FTP
- Italy [Teta Srl] WWW
- Poland [Sunsite] WWW FTP
- Portugal [lerianet] WWW FTP
- Russia [DirectNet] WWW
- Russia [IZHCOM] WWW FTP
- Russia [Scientific Center/Chernogolovka] FTP
- Romania [Timisoara] WWW FTP

-  Romania [Bucharest] WWW FTP
-  Sweden [Sunet] WWW FTP
-  Switzerland [Sunsite] WWW FTP
-  UK [Omnipotent/UK] WWW FTP
-  UK [PLiG/UK] WWW FTP
-  UK [SunSITE] WWW FTP
-  Ukraine [PACO] WWW FTP

North America:

-  Canada [Tryc] WWW
-  USA [Hurricane Electric/San Jose] WWW
-  USA [Netcasting/West Coast] FTP
-  USA [Circle Net/North Carolina] WWW
-  USA [Gina net/Florida] WWW
-  USA [pingzero/Los Angeles] WWW
-  USA [DIGEX] FTP

South America:

-  Chile [Vision] WWW

Asia:

-  China [Freecode] WWW
-  Korea [KREONet] WWW
-  Japan [Soft Agency] WWW
-  Japan [Nagoya Syouka University] WWW FTP
-  Singapore [HJC] WWW FTP
-  Taiwan [HT] WWW

Australia:

-  Australia [AARNet/Queensland] WWW FTP
-  Australia [Tas] WWW FTP
-  Australia [Blue Planet/Melbourne] WWW
-  Australia [ITworks Consulting/Victoria] WWW

Africa:

-  South−Africa [Mweb/] WWW
-  South−Africa [The Internet Solution/Johannesburg] FTP

4.2 Systèmes d'exploitation supportés par MySQL

We use GNU Autoconf so it is possible to port ***MySQL*** to all modern systems with working Posix threads and a C++ compiler. (To compile only the client code, a C++ compiler is required but not threads.) We use and develop the software ourselves primarily on Sun Solaris (versions 2.5 & 2.6) and to a lesser extent on RedHat Linux 5.0.

MySQL has been reported to compile sucessfully on the following operating system/thread package combinations. Note that for many operating systems, the native thread support works only in the latest versions.

- AIX 4.x with native threads
- BSDI 2.x with the included MIT−pthreads package
- BSDI 3.0, 3.1 and 4.x with native threads
- DEC UNIX 4.x with native threads
- FreeBSD 2.x with the included MIT−pthreads package
- FreeBSD 3.x with native threads
- HP−UX 10.20 with the included MIT−pthreads package
- HP−UX 11.x with the native threads.
- Linux 2.0+ with LinuxThreads 0.7.1 or `glibc` 2.0.7
- NetBSD 1.3/1.4 Intel and NetBSD 1.3 Alpha (Requires GNU make)
- OpenBSD 2.x with the included MIT−pthreads package
- OS/2 Warp 3, FixPack 29 and OS/2 Warp 4, FixPack 4
- SGI Irix 6.x with native threads
- Solaris 2.5, 2.6 and 2.7 with native threads on SPARC and x86
- SunOS 4.x with the included MIT−pthreads package
- SCO OpenServer with a recent port of the FSU Pthreads package
- SCO UnixWare 7.0.1
- Tru64 Unix
- Win95, Win98 and NT (the newest version is currently available only for users with a ***MySQL*** license or ***MySQL*** email support). For those who wish to test before they buy, we have released ***MySQL*** 3.21.29 (an older version) as shareware.

4.3 Quelle version de MySQL utiliser

The first decision to make is whether you want to use the latest development release or the last stable release:

- Normally, if you are beginning to use ***MySQL*** for the first time or trying to port it to some system for which there is no binary distribution, we recommend going with the development release (currently 3.22.x). This is because there are usually no really serious bugs in the development release, and you can easily test it on your machine with the `crash−me` and benchmark tests. 11 La suite de tests de MySQL.
- Otherwise, if you are running an old system and want to upgrade, but don't want to take chances with 3.22, you should upgrade to 3.21.33. We have tried to fix only fatal bugs and make small, relatively safe changes to that version.

The second decision to make is whether you want to use a source distribution or a binary distribution:

- If you want to run ***MySQL*** on a platform for which a current binary distribution exists, use that. Generally, it will be easier to install than a source distribution.
- If you want to read (and/or modify) the C and C++ code that makes up ***MySQL***, you should get a source distribution. The source code is always the ultimate manual. Source distributions also contain more tests and examples than binary distributions.

The ***MySQL*** naming scheme uses release numbers that consist of three numbers and a suffix. For example, a release name like `mysql−3.21.17−beta` is interpreted like this:

- The first number (`3`) describes the file format. All version 3 releases have the same file format. When a version 4 appears, every table will have to be converted to the new format (nice tools for this will be included, of course).
- The second number (`21`) is the release level. Normally there are two to choose from. One is the release/stable branch (currently `21`) and the other is the development branch (currently `22`) . Normally both are stable, but the development version may have quirks, missing documentation on new features or may fail to compile on some systems.
- The third number (`17`) is the version number within the release level. This is incremented for each new distribution. Usually you want the latest version for the release level you have choosen.

- The suffix (`beta`) indicates the stability level of the release. The possible suffixes are:
  - `alpha` indicates that the release contains some large section of new code that hasn't been 100% tested. Known bugs (usually there are none) should be documented in the News section. D Historique des versions de MySQL. There are also new commands and extensions in most alpha releases.
  - `beta` means that all new code has been tested. No major new features were added. There should be no known bugs.
  - `gamma` is a beta that has been around a while and seems to work fine. This is what many other companies call a release.
  - If there is no suffix, it means that the version has been run for a while at many different sites with no reports of bugs other than platform−specific bugs. This is what we call a stable release.

All versions of ***MySQL*** are run through our standard tests and benchmarks to ensure that they are relatively safe to use. Since the standard tests are extended over time to check for all previously found bugs, the test suite keeps getting better.

Note that all releases have been tested at least with:

An internal test suite
: This is part of a production system for a customer. It has many tables with hundreds of megabytes of data.

*The **MySQL** benchmark suite*
: This runs a range of common queries. It is also a test to see whether the latest batch of optimizations actually made the code faster. 11 La suite de tests de MySQL.

The `crash−me` test
: This tries to determine what features the database supports and what its capabilities and limitations are. 11 La suite de tests de MySQL.

Another test is that we use the newest ***MySQL*** version in our internal production environment, on at least one machine. We have more than 100 gigabytes of data to work with.

4.4 Comment et quand sont générées les mises à jour

MySQL is evolving quite rapidly here at TcX and we want to share this with other ***MySQL*** users. We try to make a release when we have very useful features that others seem to have a need for.

We also try to help out users who request features that are easy to implement. We also take note of what our licensed users want to have and we especially take note of what our extended email supported customers want and try to help them out.

No one has to download a new release. The News section will tell you if the new release has something you really want. D Historique des versions de MySQL.

We use the following policy when updating ***MySQL***:

- For each minor update, the last number in the version string is incremented. When there are major new features or minor incompatibilities with previous versions, the second number in the version string is incremented. When the file format changes, the first number is increased.
- Stable tested releases are meant to appear about 1−2 times a year, but if small bugs are found, a release with only bug−fixes will be released.
- Working releases are meant to appear about every 1−8 weeks.
- Binary distributions for some platforms will be made by us for major releases. Other people may make binary distributions for other systems but probably less frequently.
- We usually make patches available as soon as we have located and fixed small bugs.
- For non−critical but annoying bugs, we will make patches available if they are sent to us. Otherwise we will combine many of them into a larger patch.
- If there is, by any chance, a fatal bug in a release we will make a new release as soon as possible. We would like other companies to do this, too. :)

The current stable release is 3.22; We have already moved active development to 3.23. Bugs will still be fixed in the stable version. We don't believe in a complete freeze, as this also leaves out bug fixes and things that ``must be done''. ``Somewhat frozen'' means that we may add small things that ``almost surely will not affect anything that's already working''.

4.5 Structure de l'installation

This section describes the default layout of the directories created by installing binary and source distributions.

A binary distribution is installed by unpacking it at the installation location you choose (typically `/usr/local/mysql') and creates the following directories in that location:

Directory	*Contents of directory*
`bin'	Client programs and the `mysqld` server
`data'	Log files, databases
`include'	Include (header) files
`lib'	Libraries
`scripts'	`mysql_install_db`
`share/mysql'	Error message files
`sql−bench'	Benchmarks

A source distribution is installed after you configure and compile it. By default, the installation step installs files under `/usr/local', in the following subdirectories:

Directory	*Contents of directory*
`bin'	Client programs and scripts
`include/mysql'	Include (header) files
`info'	Documentation in Info format
`lib/mysql'	Libraries
`libexec'	The `mysqld` server
`share/mysql'	Error message files
`sql−bench'	Benchmarks and `crash−me` test
`var'	Databases and log files.

Within an installation directory, the layout of a source installation differs from that of a binary installation in the following ways:

- The `mysqld` server is installed in the `libexec' directory rather than in the `bin' directory.
- The data directory is `var' rather than `data'.
- `mysql_install_db` is installed in the `/usr/local/bin' directory rather than in `/usr/local/mysql/scripts'.
- The header file and library directories are `include/mysql' and `lib/mysql' rather than `include' and `lib'.

4.6 Installer une version binaire de MySQL

You need the following tools to install a ***MySQL*** binary distribution:

- GNU `gunzip` to uncompress the distribution.
- A reasonable `tar` to unpack the distribution. GNU `tar` is known to work.

An alternative installation méthode under Linux is to use RPM (RedHat Package Manager) distributions. 4.6.1 Linux RPM.

If you run into problems, ***PLEASE ALWAYS USE*** `mysqlbug` when posting questions to mysql@lists.mysql.com. Even if the problem isn't a bug, `mysqlbug` gathers system information that will help others solve your problem. By not using `mysqlbug`, you lessen the likelihood of getting a solution to your problem! You will find `mysqlbug` in the `` `bin' `` directory after you unpack the distribution. 2.3 Comment rapporter des bugs et des problèmes.

The basic commands you must execute to install and use a ***MySQL*** binary distribution are:

```
shell62; gunzip 60; mysql−VERSION−OS.tar.gz | tar xvf −
shell62; ln −s mysql−VERSION−OS mysql
shell62; cd mysql
shell62; scripts/mysql_install_db
shell62; bin/safe_mysqld 38;
```

You can add new users using the `bin/mysql_setpermission` script if you install the `DBI` and `Msql−Mysql−modules` Perl modules.

Here follows a more detailed description:

To install a binary distribution, follow the steps below, then proceed to 4.15 Paramètrage post−installation et tests, for post−installation setup and testing:

1. Pick the directory under which you want to unpack the distribution, and move into it. In the example below, we unpack the distribution under `` `/usr/local' `` and create a directory `` `/usr/local/mysql' `` into which ***MySQL*** is installed. (The following instructions therefore assume you have permission to create files in `` `/usr/local' ``. If that directory is protected, you will need to perform the installation as `root`.)
2. Obtain a distribution file from one of the sites listed in Getting ***MySQL***. ***MySQL*** binary distributions are provided as compressed `tar` archives and have names like `` `mysql−VERSION−OS.tar.gz' ``, where `VERSION` is a number (e.g., `3.21.15`), and `OS` indicates the type of operating system for which the distribution is intended (e.g., `pc−linux−gnu−i586`).
3. Unpack the distribution and create the installation directory:
   ```
   shell62; gunzip 60; mysql−VERSION−OS.tar.gz | tar xvf −
   shell62; ln −s mysql−VERSION−OS mysql
   ```
   The first command creates a directory named `` `mysql−VERSION−OS' ``. The second command makes a symbolic link to that directory. This lets you refer more easily to the installation directory as `` `/usr/local/mysql' ``.
4. Change into the installation directory:
   ```
   shell62; cd mysql
   ```
   You will find several files and subdirectories in the `mysql` directory. The most important for installation purposes are the `` `bin' `` and `` `scripts' `` subdirectories.

   `bin'
   This directory contains client programs and the server You should add the full pathname of this directory to your `PATH` environment variable so that your shell finds the ***MySQL*** programs properly.

   `scripts'
   This directory contains the `mysql_install_db` script used to initialize the server access permissions.
5. If you would like to use `mysqlaccess` and have the ***MySQL*** distribution in some nonstandard place, you must change the location where `mysqlaccess` expects to find the `mysql` client. Edit the `` `bin/mysqlaccess' `` script at approximately line 18. Search for a line that looks like this:
   ```
   $MYSQL     = '/usr/local/bin/mysql';    # path to mysql executable
   ```
   Change the path to reflect the location where `mysql` actually is stored on your system. If you do not do this, you will get a `broken pipe` error when you run `mysqlaccess`.
6. Create the ***MySQL*** grant tables (necessary only if you haven't installed ***MySQL*** before):
   ```
   shell62; scripts/mysql_install_db
   ```
7. If you want to install support for the Perl `DBI`/`DBD` interface, see 4.10 Remarques sur l'installation Perl.
8. If you would like ***MySQL*** to start automatically when you boot your machine, you can copy `support−files/mysql.server` to the

location where your system has its startup files. More information can be found in the `support−files/mysql.server` script itself, and in 4.15.3 Démarrer et arrête *MySQL* automatiquement.

After everything has been unpacked and installed, you should initialize and test your distribution.

You can start the ***MySQL*** server with the following command:

```
shell62; bin/safe_mysqld 38;
```

Note that ***MySQL*** versions older than 3.22.10 started the ***MySQL*** server when you run `mysql_install_db`. This is no longer true!

4.15 Paramètrage post−installation et tests.

4.6.1 Linux RPM

The recommended way to install ***MySQL*** on Linux is by using an RPM file. The ***MySQL*** RPMs are currently being built on a RedHat 5.2 system but should work on other versions of Linux that support `rpm` and use `glibc`.

If you have problems with an RPM file, for example `Sorry, the host 'xxxx' could not be looked up`, see 4.6.3.1 Linux notes.

The RPM files you may want to use are:

- `MySQL−VERSION.i386.rpm` The ***MySQL*** server. You will need this unless you only want to connect to another ***MySQL*** server running on another machine.
- `MySQL−client−VERSION.i386.rpm` The standard ***MySQL*** client programs. You probably always want to install this package.
- `MySQL−bench−VERSION.i386.rpm` Tests and benchmarks. Requires Perl and msql−mysql−modules RPMs.
- `MySQL−devel−VERSION.i386.rpm` Libraries and include files needed if you want to compile other ***MySQL*** clients, such as the Perl modules.
- `MySQL−VERSION.src.rpm` This contains the source code for all of the above packages. It can also be used to try to build RPMs for other architectures (for example, Alpha or SPARC).

To see all files in an RPM package:

```
shell62; rpm −qpl MySQL−VERSION.i386.rpm
```

To perform a standard minimal installation, run this command:

```
shell62; rpm −i MySQL−VERSION.i386.rpm MySQL−client−VERSION.i386.rpm
```

To install just the client package:

```
shell62; rpm −i MySQL−client−VERSION.i386.rpm
```

The RPM places data in `` `/var/lib/mysql' ``. The RPM also creates the appropriate entries in `` `/sbin/rc.d/' `` to start the server automatically at boot time. (This means that if you have performed a previous installation, you may want to make a copy of your previously−installed ***MySQL*** startup file if you made any changes to it, so you don't lose your changes.)

After installing the RPM file(s), go to the binary install section and use the instructions there, starting from the step that creates the ***MySQL*** grant tables. 4.6 Installer une version binaire de MySQL.

4.6.2 Construire un programme client

If you compile ***MySQL*** clients that you've written yourself or that you obtain from a third party, they must be linked using the `−lmysqlclient` option on the link command. You may also need to specify a `−L` option to tell the linker where to find the library. For example, if the library is installed in `` `/usr/local/mysql/lib' ``, use `−L/usr/local/mysql/lib −lmysqlclient` on the link command.

For clients that use ***MySQL*** header files, you may need to specify a `−I` option when you compile them (for example, `−I/usr/local/mysql/include`), so the compiler can find the header files.

4.6.3 Instruction spécifiques aux OS

The following sections indicate some of the issues that have been observed to occur on particular systems when installing ***MySQL*** from a binary distribution.

4.6.3.1 Linux notes

MySQL needs at least Linux 2.0.

The binary release is linked with `−static`, which means you not normally need not worry about which version of the system libraries you have. You need not install LinuxThreads, either. A program linked with `−static` is slightly bigger than a dynamically−linked program but also slightly faster (3−5%). One problem however is that you can't use user definable functions (UDFs) with a statically−linked program. If you are going to write or use UDF functions (this is something only for C or C++ programmers) you must compile ***MySQL*** yourself, using dynamic linking.

If you are using a `libc`−based system (instead of a `glibc2` system), you will probably get some problems with hostname resolving and getpwnam() with the binary release. (This is because `glibc` unfortunately depends on some external libraries to resolve hostnames and getwpent() , even when compiled with `−static`). In this case you probably get the following error message when you run `mysql_install_db`:

```
Sorry, the host 'xxxx' could not be looked up
```

or the following error when you try to run mysqld with the `−−user` option:

```
getpwnam: No such file or directory
```

You can solve this problem one of the following ways:

- Get a ***MySQL*** source distribution (an RPM or the `tar` distribution) and install this instead.
- Execute `mysql_install_db −−force`; This will not execute the `resolveip` test in `mysql_install_db`. The downside is that you can't use host names in the grant tables; you must use IP numbers instead (except for `localhost`). If you are using an old ***MySQL*** release that doesn't support `−−force` you have to remove the `resolveip` test in `mysql_install` with an editor.
- Start mysqld with `su` instead of using `−−user`.

The Linux−Intel binary and RPM releases of ***MySQL*** are configured for the highest possible speed. We are always trying to use the fastest stable compiler available.

MySQL Perl support requires Perl 5.004_03 or newer.

4.6.3.2 HP−UX notes

The binary distribution of ***MySQL*** for HP−UX is distributed as an HP depot file and as a tar file. To use the depot file you must be running at least HP−UX 10.x to have access to HP's software depot tools.

The HP version of ***MySQL*** was compiled on an HP 9000/8xx server under HP−UX 10.20, and uses MIT−pthreads. It is known to work well under this configuration. ***MySQL*** 3.22.26 and newer can also be built with HP's native thread package.

Other configurations that may work:

- HP 9000/7xx running HP−UX 10.20+
- HP 9000/8xx running HP−UX 10.30

The following configurations almost definitely won't work:

- HP 9000/7xx or 8xx running HP−UX 10.x where x < 2
- HP 9000/7xx or 8xx running HP−UX 9.x

To install the distribution, use one of the commands below, where `/path/to/depot` is the full pathname of the depot file:

- To install everything, including the server, client and development tools:
```
shell62; /usr/sbin/swinstall −s /path/to/depot mysql.full
```
- To install only the server:
```
shell62; /usr/sbin/swinstall −s /path/to/depot mysql.server
```
- To install only the client package:
```
shell62; /usr/sbin/swinstall −s /path/to/depot mysql.client
```
- To install only the development tools:
```
shell62; /usr/sbin/swinstall −s /path/to/depot mysql.developer
```

The depot places binaries and libraries in `` `/opt/mysql' `` and data in `` `/var/opt/mysql' ``. The depot also creates the appropriate entries in `` `/sbin/init.d' `` and `` `/sbin/rc2.d' `` to start the server automatically at boot time. Obviously, this entails being `root` to install.

To install the HP−UX tar distribution, you must have a copy of gnu tar.

4.7 Installer une version source de MySQL

You need the following tools to build and install ***MySQL*** from source:

- GNU `gunzip` to uncompress the distribution.
- A reasonable `tar` to unpack the distribution. GNU `tar` is known to work.
- A working ANSI C++ compiler. `gcc` >= 2.8.1, `egcs` >= 1.0.2, SGI C++ and SunPro C++ are some of the compilers that are known to work. `libg++` is not needed when using `gcc`. `gcc` 2.7.x has a bug that makes it impossible to compile some perfectly legal C++ files, such as `` `sql/sql_base.cc' ``. If you only have `gcc` 2.7.x, you must upgrade your `gcc` to be able to compile ***MySQL***.
- A good `make` program. GNU `make` is always recommended and is sometimes required. If you have problems, we recommend trying GNU `make` 3.75 or newer.

If you run into problems, ***PLEASE ALWAYS USE `mysqlbug`*** when posting questions to mysql@lists.mysql.com. Even if the problem isn't a bug, `mysqlbug` gathers system information that will help others solve your problem. By not using `mysqlbug`, you lessen the likelihood of getting a solution to

your problem! You will find `mysqlbug` in the `` `scripts' `` directory after you unpack the distribution. 2.3 Comment rapporter des bugs et des problèmes.

4.7.1 Introduction à l'installation rapide

The basic commands you must execute to install a ***MySQL*** source distribution are (from an unpacked `tar` file):

```
shell62; configure
shell62; make
shell62; make install
shell62; scripts/mysql_install_db
shell62; /usr/local/mysql/bin/safe_mysqld 38;
```

If you start from a source RPM, then do the following.

```
shell62; rpm −−rebuild MySQL−VERSION.src.rpm
```

This will make a binary RPM that you can install.

You can add new users using the `bin/mysql_setpermission` script if you install the `DBI` and `Msql−Mysql−modules` Perl modules.

Here follows a more detailed description:

To install a source distribution, follow the steps below, then proceed to 4.15 Paramètrage post−installation et tests, for post−installation initialization and testing.

1. Pick the directory under which you want to unpack the distribution, and move into it.
2. Obtain a distribution file from one of the sites listed in Getting ***MySQL***. ***MySQL*** source distributions are provided as compressed `tar` archives and have names like `` `mysql−VERSION.tar.gz' ``, where `VERSION` is a number like 3.23.5−alpha.
3. Unpack the distribution into the current directory:
   ```
   shell62; gunzip 60; mysql−VERSION.tar.gz | tar xvf −
   ```
   This command creates a directory named `` `mysql−VERSION' ``.
4. Change into the top−level directory of the unpacked distribution:
   ```
   shell62; cd mysql−VERSION
   ```
5. Configure the release and compile everything:
   ```
   shell62; ./configure −−prefix=/usr/local/mysql
   shell62; make
   ```
   When you run `configure`, you might want to specify some options. Run `./configure −−help` for a list of options. `configure` options, discusses some of the more useful options. If `configure` fails, and you are going to send mail to lines from `` `config.log' `` that you think can help solve the problem. Also include the last couple of lines of output from `configure` if `configure` aborts. Post the bug report using the `mysqlbug` script. 2.3 Comment rapporter des bugs et des problèmes. If the compile fails, see 4.8 Problèmes de compilation?, for help with a number of common problems.
6. Install everything:
   ```
   shell62; make install
   ```
   You might need to run this command as `root`.
7. Create the ***MySQL*** grant tables (necessary only if you haven't installed ***MySQL*** before):
   ```
   shell62; scripts/mysql_install_db
   ```

Note that ***MySQL*** versions older than 3.22.10 started the ***MySQL*** server when you run `mysql_install_db`. This is no longer true!

8. If you want to install support for the Perl `DBI`/`DBD` interface, see 4.10 Remarques sur l'installation Perl.
9. If you would like ***MySQL*** to start automatically when you boot your machine, you can copy `support−files/mysql.server` to the location where your system has its startup files. More information can be found in the `support−files/mysql.server` script itself, and in 4.15.3 Démarrer et arrête ***MySQL*** automatiquement.

After everything has been installed, you should initialize and test your distribution.

You can start the ***MySQL*** server with the following command, where `BINDIR` is the directory in which `safe_mysqld` is installed (`/usr/local/bin' by default):

```
shell62; BINDIR/safe_mysqld 38;
```

If that command fails immediately with `mysqld daemon ended` then you can find some information in the file `mysql−data−directory/'hostname'.err'. The likely reason is that you already have another `mysqld` server running. 19.3 Faire tourner plusieurs serveurs MySQL sur la même machine.

4.15 Paramètrage post−installation et tests.

4.7.2 Appliquer un patch

Sometimes patches appear on the mailing list or are placed in the patches area of the ***MySQL*** FTP site.

To apply a patch from the mailing list, save the message in which the patch appears in a file, change into the top−level directory of your ***MySQL*** source tree and run these commands:

```
shell62; patch −p1 60; patch−file−name
shell62; rm config.cache
shell62; make clean
```

Patches from the FTP site are distributed as plain text files or as files compressed with `gzip` files. Apply a plain patch as shown above for mailing list patches. To apply a compressed patch, change into the top−level directory of your ***MySQL*** source tree and run these commands:

```
shell62; gunzip 60; patch−file−name.gz | patch −p1
shell62; rm config.cache
shell62; make clean
```

After applying a patch, follow the instructions for a normal source install, beginning with the `./configure` step. After running the `make install` step, restart your ***MySQL*** server.

You may need to bring down any currently running server before you run `make install`. (Use `mysqladmin shutdown` to do this.) Some systems do not allow you to install a new version of a program if it replaces the version that is currently executing.

4.7.3 Options communes de `configure`

The `configure` script gives you a great deal of control over how you configure your ***MySQL*** distribution. Typically you do this using options on the `configure` command line. You can also affect `configure` using certain environment variables. For a list of options supported by `configure`, run this command:

```
shell62; ./configure --help
```

Some of the more commonly−used `configure` options are described below:

- To compile just the ***MySQL*** client libraries and client programs and not the server, use the `--without-server` option:
  ```
  shell62; ./configure --without-server
  ```

  If you don't have a C++ compiler, `mysql` will not compile (it is the one client program that requires C++). In this case, you can remove the code in `configure` that tests for the C++ compiler and then run `./configure` with the `--without-server` option. The compile step will still try to build `mysql`, but you can ignore any warnings about `` `mysql.cc' ``. (If `make` stops, try `make -k` to tell it to continue with the rest of the build even if errors occur.)

- If you don't want your log files and database directories located under `` `/usr/local/var' ``, use a `configure` command something like one of these:
  ```
  shell62; ./configure --prefix=/usr/local/mysql
  shell62; ./configure --prefix=/usr/local \
             --localstatedir=/usr/local/mysql/data
  ```

  The first command changes the installation prefix so that everything is installed under `` `/usr/local/mysql' `` rather than the default of `` `/usr/local' ``. The second command preserves the default installation prefix, but overrides the default location for database directories (normally `` `/usr/local/var' ``) and changes it to `/usr/local/mysql/data`.

- If you are using Unix and you want the ***MySQL*** socket located somewhere other than the default location (normally in the directory `` `/tmp' `` or `` `/var/run' ``, use a `configure` command like this:
  ```
  shell62; ./configure --with-unix-socket-path=/usr/local/mysql/tmp/mysql.sock
  ```

  Note that the given file must be an absolute pathname!

- If you want to compile statically−linked programs (e.g., to make a binary distribution, to get more speed or to work around problems with some RedHat distributions), run `configure` like this:
  ```
  shell62; ./configure --with-client-ldflags=-all-static \
             --with-mysqld-ldflags=-all-static
  ```

- If you are using `gcc` and don't have `libg++` or `libstdc++` installed, you can tell `configure` to use `gcc` as your C++ compiler:
  ```
  shell62; CC=gcc CXX=gcc ./configure
  ```

  When you use `gcc` as your C++ compiler, it will not attempt to link in `libg++` or `libstdc++`. If the build fails and produces errors about your compiler or linker not being able to create the shared library `` `libmysqlclient.so.#' `` (`` `#' `` is a version number), you can work around this problem by giving the `--disable-shared` option to `configure`. In this case, `configure` will not build a shared `libmysqlclient.so.#` library.

- You can configure ***MySQL*** not to use `DEFAULT` column values for non−`NULL` columns (i.e., columns that are not allowed to be `NULL`). This causes `INSERT` statements to generate an error unless you explicitly specify values for all columns that require a non−`NULL` value. To suppress use of default values, run `configure` like this:
  ```
  shell62; CXXFLAGS=-DDONT_USE_DEFAULT_FIELDS ./configure
  ```

- By default, ***MySQL*** uses the ISO−8859−1 (Latin1) character set. To change the default set, use the `--with-charset` option:
  ```
  shell62; ./configure --with-charset=CHARSET
  ```

  `CHARSET` may be one of `big5`, `cp1251`, `cp1257`, `czech`, `danish`,`dec8`, `dos`, `euc_kr`, `german1`, `hebrew`, `hp8`, `hungarian`, `koi8_ru`, `koi8_ukr`, `latin1`, `latin2`, `sjis`, `swe7`, `tis620`, `ujis`, `usa7`, `win1251` or `win1251ukr`. 9.1.1 Le jeu de caractère utilisé pour le tri des données. Note that if you want to change the character set, you must do a `make distclean` between configurations! If you want to convert characters between the server and the client, you should take a look at the `SET OPTION CHARACTER SET` command. `SET OPTION`. ***Warning:*** If you change character sets after having created any tables, you will have to run `isamchk -r -q` on every table. Your indexes may be sorted incorrectly otherwise. (This can happen if you install ***MySQL***, create some tables, then reconfigure ***MySQL*** to use a different character set and reinstall it.)

- To configure ***MySQL*** with debugging code, use the `--with-debug` option:
  ```
  shell62; ./configure --with-debug
  ```

  This causes a safe memory allocator to be included that can find some errors and that provides output about what is happening. G.1 Debugguer un serveur MySQL.

- Options that pertain to particular systems can be found in the system−specific sections later in this chapter. 4.11 Quelques spécificités liées aux OS.

4.8 Problèmes de compilation?

All ***MySQL*** programs compile cleanly for us with no warnings on Solaris using `gcc`. On other systems, warnings may occur due to differences in system include files. See 4.9 Remarques sur MIT−pthreads, for warnings that may occur when using MIT−pthreads. For other problems, check the list below.

The solution to many problems involves reconfiguring. If you do need to reconfigure, take note of the following:

- If `configure` is run after it already has been run, it may use information that was gathered during its previous invocation. This information is stored in `` `config.cache' ``. When `configure` starts up, it looks for that file and reads its contents if it exists, on the assumption that the information is still correct. That assumption is invalid when you reconfigure.
- Each time you run `configure`, you must run `make` again to recompile. However, you may want to remove old object files from previous builds first, since they were compiled using different configuration options.

To prevent old configuration information or object files from being used, run these commands before rerunning `configure`:

```
shell62; rm config.cache
shell62; make clean
```

Alternatively, you can run `make distclean`.

The list below describes some of the problems compiling ***MySQL*** that have been found to occur most often:

- If you get errors when compiling `` `sql_yacc.cc' `` such as the ones shown below, you have probably run out of memory or swap space:
  ```
  Internal compiler error: program cc1plus got fatal signal 11
    or
  Out of virtual memory
    or
  Virtual memory exhausted
  ```

  The problem is that `gcc` requires huge amounts of memory to compile `` `sql_yacc.cc' `` with inline functions. Try running `configure` with the `--with−low−memory` option:

  ```
  shell62; ./configure --with−low−memory
  ```

  This option causes `−fno−inline` to be added to the compile line if you are using `gcc` and `−O0` if you are using something else. You should try the `--with−low−memory` option even if you have so much memory and swap space that you think you can't possibly have run out. This problem has been observed to occur even on systems with generous hardware configurations, and the `--with−low−memory` option usually fixes it.

- By default, `configure` picks `c++` as the compiler name and GNU `c++` links with `−lg++`. If you are using `gcc`, that behavior can cause problems during configuration such as this:
  ```
  configure: error: installation or configuration problem:
  C++ compiler cannot create executables.
  ```

  You might also observe problems during compilation related to `g++`, `libg++` or `libstdc++`. One cause of these problems is that you may not have `g++`, or you may have `g++` but not `libg++` or `libstdc++`. Take a look at the `` `config.log' `` file. It should contain the exact reason why your c++ compiler didn't work! To work around these problems, you can use `gcc` as your C++ compiler. Try setting the environment variable `CXX` to `"gcc −O3"`. For example:

  ```
  shell62; CXX="gcc −O3" ./configure
  ```

This works because `gcc` compiles C++ sources as well as `g++` does, but does not link in `libg++` or `libstdc++` by default. Another way to fix these problems, of course, is to install `g++`, `libg++` and `libstdc++`.

- If your compile fails with errors such as any of the following, you must upgrade your version of `make` to GNU `make`:

```
making all in mit-pthreads
make: Fatal error in reader: Makefile, line 18:
Badly formed macro assignment
  or
make: file `Makefile' line 18: Must be a separator (:
  or
pthread.h: No such file or directory
```

Solaris and FreeBSD are known to have troublesome `make` programs. GNU `make` version 3.75 is known to work.

- If you want to define flags to be used by your C or C++ compilers, do so by adding the flags to the `CFLAGS` and `CXXFLAGS` environment variables. You can also specify the compiler names this way using `CC` and `CXX`. For example:

```
shell62; CC=gcc
shell62; CFLAGS=-O6
shell62; CXX=gcc
shell62; CXXFLAGS=-O6
shell62; export CC CFLAGS CXX CXXFLAGS
```

See 4.14 TcX, for a list of flag definitions that have been found to be useful on various systems.

- If you get an error message like this, you need to upgrade your `gcc` compiler:

```
client/libmysql.c:273: parse error before `__attribute__'
```

`gcc` 2.8.1 is known to work, but we recommend using `egcs` 1.0.3a or newer instead.

- If you get errors such as those shown below when compiling `mysqld`, `configure` didn't correctly detect the type of the last argument to `accept()`, `getsockname()` or `getpeername()`:

```
cxx: Error: mysqld.cc, line 645: In this statement, the referenced
     type of the pointer value "38;length" is "unsigned long", which
     is not compatible with "int".
new_sock = accept(sock, (struct sockaddr *) 38;length);
```

To fix this, edit the `config.h' file (which is generated by `configure`). Look for these lines:

```
/* Define as the base type of the last arg to accept */
#define SOCKET_SIZE_TYPE XXX
```

Change `XXX` to `size_t` or `int`, depending on your operating system. (Note that you will have to do this each time you run `configure`, since `configure` regenerates `config.h'.)

- The `sql_yacc.cc' file is generated from `sql_yacc.yy'. Normally the build process doesn't need to create `sql_yacc.cc', because ***MySQL*** comes with an already−generated copy. However, if you do need to recreate it, you might encounter this error:

```
"sql_yacc.yy", line xxx fatal: default action causes potential...
```

This is a sign that your version of `yacc` is deficient. You probably need to install `bison` (the GNU version of `yacc`) and use that instead.

- If you need to debug `mysqld` or a ***MySQL*** client, run `configure` with the `--with-debug` option, then recompile and link your clients with the new client library. G.2 Debugguer un client MySQL.

4.9 Remarques sur MIT−pthreads

This section describes some of the issues involved in using MIT−pthreads.

Note that on Linux you should NOT use MIT−pthreads but install LinuxThreads! 4.11.5 Linux (Toutes versions de Linux).

If your system does not provide native thread support, you will need to build ***MySQL*** using the MIT−pthreads package. This includes most FreeBSD systems, SunOS 4.x, Solaris 2.4 and earlier, and some others. 4.2 Systèmes d'exploitation supportés par MySQL.

- On most systems, you can force MIT−pthreads to be used by running `configure` with the `--with-mit-threads` option:
```
shell62; ./configure --with-mit-threads
```

  Building in a non−source directory is not supported when using MIT−pthreads, because we want to minimize our changes to this code.

- MIT−pthreads doesn't support the `AF_UNIX` protocol used to implement Unix sockets. This means that if you compile using MIT−pthreads, all connections must be made using TCP/IP (which is a little slower). If you find after building ***MySQL*** that you cannot connect to the local server, it may be that your client is attempting to connect to `localhost` using a Unix socket as the default. Try making a TCP/IP connection with `mysql` by using a host option (`-h` or `--host`) to specify the local host name explicitly.
- The checks that determine whether or not to use MIT−pthreads occur only during the part of the configuration process that deals with the server code. If you have configured the distribution using `--without-server` to build only the client code, clients will not know whether or not MIT−pthreads is being used and will use Unix socket connections by default. Since Unix sockets do not work under MIT−pthreads, this means you will need to use `-h` or `--host` when you run client programs.
- When ***MySQL*** is compiled using MIT−pthreads, system locking is disabled by default for performance reasons. You can tell the server to use system locking with the `--use-locking` option.
- Sometimes the pthread `bind()` command fails to bind to a socket without any error message (at least on Solaris). The result is that all connections to the server fail. For example:
```
shell62; mysqladmin version
mysqladmin: connect to server at '' failed;
error: 'Can't connect to mysql server on localhost (146)'
```

  The solution to this is to kill the `mysqld` server and restart it. This has only happened to us when we have forced the server down and done a restart immediately.

- With MIT−pthreads, the `sleep()` system call isn't interruptible with `SIGINT` (break). This is only noticeable when you run `mysqladmin --sleep`. You must wait for the `sleep()` call to terminate before the interrupt is served and the process stops.
- When linking you may receive warning messages like these (at least on Solaris); they can be ignored:
```
ld: warning: symbol `_iob' has differing sizes:
    (file /my/local/pthreads/lib/libpthread.a(findfp.o) value=0x4;
file /usr/lib/libc.so value=0x140);
    /my/local/pthreads/lib/libpthread.a(findfp.o) definition taken
ld: warning: symbol `__iob' has differing sizes:
    (file /my/local/pthreads/lib/libpthread.a(findfp.o) value=0x4;
file /usr/lib/libc.so value=0x140);
    /my/local/pthreads/lib/libpthread.a(findfp.o) definition taken
```

- Some other warnings also can be ignored:
```
implicit declaration of function `int strtoll(...)'
implicit declaration of function `int strtoul(...)'
```

- We haven't gotten `readline` to work with MIT−pthreads. (This isn't needed, but may be interesting for someone.)

4.10 Remarques sur l'installation Perl

4.10.1 Installer Perl sous Unix

Perl support for ***MySQL*** is provided by means of the `DBI`/`DBD` client interface. 20.5 API MySQL en Perl. The Perl `DBD`/`DBI` client code requires Perl 5.004 or later. The interface *will not work* if you have an older version of Perl.

MySQL Perl support also requires that you've installed ***MySQL*** client programming support. If you installed ***MySQL*** from RPM files, client programs are in the client RPM, but client programming support is in the developer RPM. Make sure you've installed the latter RPM.

As of release 3.22.8, Perl support is distributed separately from the main ***MySQL*** distribution. If you want to install Perl support, the files you will need can be obtained from http://www.mysql.com/Contrib.

The Perl distributions are provided as compressed `tar` archives and have names like `` `MODULE−VERSION.tar.gz' ``, where `MODULE` is the module name and `VERSION` is the version number. You should get the `Data−Dumper`, `DBI`, and `Msql−Mysql−modules` distributions and install them in that order. The installation procedure is shown below. The example shown is for the `Data−Dumper` module, but the procedure is the same for all three distributions.

1. Unpack the distribution into the current directory:
   ```
   shell62; gunzip 60; Data−Dumper−VERSION.tar.gz | tar xvf −
   ```
   This command creates a directory named `` `Data−Dumper−VERSION' ``.

2. Change into the top−level directory of the unpacked distribution:
   ```
   shell62; cd Data−Dumper−VERSION
   ```

3. Build the distribution and compile everything:
   ```
   shell62; perl Makefile.PL
   shell62; make
   shell62; make test
   shell62; make install
   ```

The `make test` command is important, because it verifies that the module is working. Note that when you run that command during the `Msql−Mysql−modules` installation to exercise the interface code, the ***MySQL*** server must be running or the test will fail.

It is a good idea to rebuild and reinstall the `Msql−Mysql−modules` distribution whenever you install a new release of ***MySQL***, particularly if you notice symptoms such as all your `DBI` scripts dumping core after you upgrade ***MySQL***.

If you don't have the right to install Perl modules in the system directory or if you to install local Perl modules, the following reference may help you:

`http://www.iserver.com/support/contrib/perl5/modules.html`

Look under the heading `Installing New Modules that Require Locally Installed Modules`.

4.10.2 Installer ActiveState Perl sous Win32

To install the ***MySQL*** `DBD` module with ActiveState Perl on Win32, you should do the following:

- Open a DOS shell.
- If required, set the HTTP_proxy variable. For example, you might try: `set HTTP_proxy=my.proxy.com:3128`
- Start the PPM program: `C:\perl\bin\ppm.pl`
- If you have not already done so, install `DBI`: `install DBI`
- If this succeeds, install `DBD::mysql`: `http://www.mysql.com/Contrib/ppd/DBD−mysql.ppd`

If you can't get the above to work, you should instead install the ***MyODBC*** driver and connect to ***MySQL*** server through ODBC.

```
use DBI;
$dbh= DBI−62;connect("DBI:ODBC:$dsn","$user","$password") ||
  die "Got error $DBI::errstr when connecting to $dsn\n";
```

4.10.3 Installer la distribution Perl de *MySQL* sous Win32

The ***MySQL*** Perl distribution contains `DBI`, `DBD:MySQL` and `DBD:ODBC`.

- Get the Perl distribution for Win32 from http://www.mysql.com/download.html.
- Unzip the distribution in `C:` so that you get a `` `C:\PERL' `` directory.
- Add the directory `` `C:\PERL\BIN' `` to your path.
- Add the directory `` `C:\PERL\BIN\MSWin32−x86−thread' `` or `` `C:\PERL\BIN\MSWin32−x86' `` to your path.
- Test that `perl` works by executing `perl −v` in a DOS shell.

4.10.4 Problèmes avec l'interface `DBI/DBD` de Perl

If Perl reports that it can't find the `../mysql/mysql.so` module, then the problem is probably that Perl can't locate the shared library `` `libmysqlclient.so' ``.

You can fix this by any of the following méthodes:

- Compile the `Msql−Mysql−modules` distribution with `perl Makefile.PL −static` rather than `perl Makefile.PL`
- Copy `libmysqlclient.so` to the directory where your other shared libraries are located (probably `` `/usr/lib' `` or `` `/lib' ``).
- On `Linux` you can add the pathname of the directory where `libmysqlclient.so` is located to the `` `/etc/ld.so.conf' `` file.
- Add the pathname of the directory where `libmysqlclient.so` is located to the `LD_RUN_PATH` environment variable.

If you get the following errors from `DBD−mysql`, you are probably using `gcc` (or using an old binary compiled with `gcc`):

```
/usr/bin/perl: can't resolve symbol '__moddi3'
/usr/bin/perl: can't resolve symbol '__divdi3'
```

Add `−L/usr/lib/gcc−lib/... −lgcc` to the link command when the `` `mysql.so' `` library gets built (check the output from `make` for `` `mysql.so' `` when you compile the Perl client). The `−L` option should specify the pathname of the directory where `` `libgcc.a' `` is located on your system.

Another cause of this problem may be that Perl and ***MySQL*** aren't both compiled with `gcc`. In this case, you can solve the mismatch by compiling both with `gcc`.

If you want to use the Perl module on a system that doesn't support dynamic linking (like SCO) you can generate a static version of Perl that includes `DBI` and `DBD−mysql`. The way this works is that you generate a version of Perl with the `DBI` code linked in and install it on top of your current Perl. Then you use that to build a version of Perl that additionally has the `DBD` code linked in, and install that.

On SCO, you must have the following environment variables set:

```
shell62; LD_LIBRARY_PATH=/lib:/usr/lib:/usr/local/lib:/usr/progressive/lib
or
shell62; LD_LIBRARY_PATH=/usr/lib:/lib:/usr/local/lib:/usr/ccs/lib:/usr/progressive/lib:/usr/skun
shell62; LIBPATH=/usr/lib:/lib:/usr/local/lib:/usr/ccs/lib:/usr/progressive/lib:/usr/skunk/lib
shell62; MANPATH=scohelp:/usr/man:/usr/local1/man:/usr/local/man:/usr/skunk/man:
```

First, create a Perl that includes a statically−linked `DBI` by running these commands in the directory where your `DBI` distribution is located:

```
shell62; perl Makefile.PL LINKTYPE=static
shell62; make
shell62; make install
shell62; make perl
```

Then you must install the new Perl. The output of `make perl` will indicate the exact `make` command you will need to execute to perform the installation. On SCO, this is `make −f Makefile.aperl inst_perl MAP_TARGET=perl`.

Next, use the just−created Perl to create another Perl that also includes a statically−linked `DBD::mysql` by running these commands in the directory where your `Msql−Mysql−modules` distribution is located:

```
shell62; perl Makefile.PL LINKTYPE=static
shell62; make
shell62; make install
shell62; make perl
```

Finally, you should install this new Perl. Again, the output of `make perl` indicates the command to use.

4.11 Quelques spécificités liées aux OS

The following sections indicate some of the issues that have been observed to occur on particular systems when installing ***MySQL*** from a source distribution.

4.11.1 Solaris

On Solaris, you may run into trouble even before you get the ***MySQL*** distribution unpacked! Solaris `tar` can't handle long file names, so you may see an error like this when you unpack ***MySQL***:

```
x mysql−3.22.12−beta/bench/Results/ATIS−mysql_odbc−NT_4.0−cmp−db2,
informix,ms−sql,mysql,oracle,solid,sybase, 0 bytes, 0 tape blocks
tar: directory checksum error
```

In this case, you must use GNU `tar` (`gtar`) to unpack the distribution. You can find a precompiled copy for Solaris at http://www.mysql.com/Downloads/.

Sun native threads work only on Solaris 2.5 and higher. For 2.4 and earlier versions, ***MySQL*** will automatically use MIT−pthreads. 4.9 Remarques sur MIT−pthreads.

If you get the following error from configure:

```
checking for restartable system calls... configure: error can not run test
programs while cross compiling
```

This means that you have something wrong with your compiler installation! In this case you should upgrade your compiler to a newer version. You may also be able to solve this problem by inserting the following row into the `config.cache` file:

```
ac_cv_sys_restartable_syscalls=${ac_cv_sys_restartable_syscalls='no'}
```

If you are using Solaris on a SPARC, the recommended compiler is `egcs` 1.1.2 or newer. You can find this at http://egcs.cygnus.com/. Note that `egs` 1.1.1 and `gcc` 2.8.1 don't work reliably on SPARC!

The recommended `configure` line when using `egcs` 1.1.2 is:

```
shell62; CC=gcc CFLAGS="−O6"
       CXX=gcc CXXFLAGS="−O6 −felide−constructors −fno−exceptions −fno−rtti"
```

```
./configure −−prefix=/usr/local/mysql −−with−low−memory
```

If you have the Sun Workshop 4.2 compiler, you can run `configure` like this:

CC=cc CFLAGS="−xstrconst −Xa −xO4 −native −mt" CXX=CC CXXFLAGS="−xO4 −native −noex −mt" ./configure −−prefix=/usr/local/mysql

```
shell62; CC=cc CFLAGS="−Xa −fast −xO4 −native −xstrconst −mt" \
       CXX=CC CXXFLAGS="−noex −XO4 −mt" \
       ./configure
```

You may also have to edit the `configure` script to change this line:

```
#if !defined(__STDC__) || __STDC__ != 1
```

to this:

```
#if !defined(__STDC__)
```

If you turn on `__STDC__` with the `−Xc` option, the Sun compiler can't compile with the Solaris `` `pthread.h' `` header file. This is a Sun bug (broken compiler or broken include file).

If `mysqld` issues the error message shown below when you run it, you have tried to compile ***MySQL*** with the Sun compiler without enabling the multi−thread option (`−mt`):

```
libc internal error: _rmutex_unlock: rmutex not held
```

Add `−mt` to `CFLAGS` and `CXXFLAGS` and try again.

If you get the following error when compiling ***MySQL*** with `gcc`, it means that your `gcc` is not configured for your version of Solaris!

```
shell62; gcc −O3 −g −O2 −DDBUG_OFF  −o thr_alarm ...
./thr_alarm.c: In function `signal_hand':
./thr_alarm.c:556: too many arguments to function `sigwait'
```

The proper thing to do in this case is to get the newest version of `egcs` and compile it with your current `gcc` compiler! At least for Solaris 2.5, almost all binary versions of `gcc` have old, unusable include files that will break all programs that use threads (and possibly other programs)!

Solaris doesn't provide static versions of all system libraries (`libpthreads` and `libdl`), so you can't compile ***MySQL*** with `−−static`. If you try to do so, you will get the error:

```
ld: fatal: library −ldl: not found
```

If too many processes try to connect very rapidly to `mysqld`, you will see this error in the ***MySQL*** log:

```
Error in accept: Protocol error
```

You might try starting the server with the `−−set−variable back_log=50` option as a workaround for this.

If you are linking your own ***MySQL*** client, you might get the following error when you try to execute it:

```
ld.so.1: ./my: fatal: libmysqlclient.so.#: open failed: No such file or directory
```

The problem can be avoided by one of the following méthodes:

- Link the client with the following flag (instead of `-Lpath`): `-Wl,r/full-path-to-libmysqlclient.so`.
- Copy `libmysqclient.so` to `/usr/lib'.
- Add the pathname of the directory where `libmysqlclient.so` is located to the `LD_RUN_PATH` environment variable before running your client.

4.11.2 Solaris 2.7

You can normally use a Solaris 2.6 binary on Solaris 2.7. Most of the Solaris 2.6 issues also apply for Solaris 2.7.

Solaris 2.7 has some bugs in the include files. You may see the following error when you use `gcc`:

```
/usr/include/widec.h:42: warning: `getwc' redefined
/usr/include/wchar.h:326: warning: this is the location of the previous
definition
```

If this occurs, you can do the following to fix the problem:

Copy `/usr/include/widec.h` to `.../lib/gcc-lib/os/gcc-version/include` and change line 41 from:

```
#if     !defined(lint) 38;!defined(__lint)

to

#if     !defined(lint) 38;!defined(__lint) 38;!defined(getwc)
```

Alternatively, you can edit `/usr/include/widec.h' directly. Either way, after you make the fix, you should remove `config.cache' and run `configure` again!

If you get errors like this when you run `make`, it's because `configure` didn't detect the `curses.h' file (probably because of the error in `/usr/include/widec.h`:

```
In file included from mysql.cc:50:
/usr/include/term.h:1060: syntax error before `,'
/usr/include/term.h:1081: syntax error before `;'
```

The solution to this is to do one of the following steps:

- Edit `/usr/include/widec.h' as indicted above and rerun configure
- Remove the `#define HAVE_TERM` line from `config.h' file and run `make` again.
- Configure with `CFLAGS=-DHAVE_CURSES CXXFLAGS=-DCURSES ./configure`

4.11.3 Solaris x86

If you are using `gcc` or `egcs` on Solaris x86 and you experience problems with core dumps under load, you should use the following `configure` command:

```
shell62; CC=gcc CFLAGS="-O6 -fomit-frame-pointer" \
       CXX=gcc \
       CXXFLAGS="-O6 -fomit-frame-pointer -felide-constructors -fno-exceptions -fno-rtti" \
```

```
./configure --prefix=/usr/local/mysql
```

This will avoid problems with the `libstdc++` library and with C++ exceptions.

If this doesn't help, you should compile a debug version and run it with a trace file or under `gdb`. G.1 Debugguer un serveur MySQL.

4.11.4 SunOS 4

On SunOS 4, MIT−pthreads is needed to compile ***MySQL***, which in turn means you will need GNU `make`.

Some SunOS 4 systems have problems with dynamic libraries and `libtool`. You can use the following `configure` line to avoid this problem:

```
shell62; ./configure --disable-shared --with-mysqld-ldflags=-all-static
```

When compiling `readline`, you may get warnings about duplicate defines. These may be ignored.

When compiling `mysqld`, there will be some `implicit declaration of function` warnings. These may be ignored.

4.11.5 Linux (Toutes versions de Linux)

MySQL uses LinuxThreads on Linux. If you are using an old Linux version that doesn't have `glibc2`, you must install LinuxThreads before trying to compile ***MySQL***. http://www.mysql.com/Downloads/Linux

If you can't start `mysqld` or if `mysql_install_db` doesn't work, please continue reading! This only happens on Linux system with problems in the LinuxThreads or `libc`/`glibc` libraries. There are a lot of simple workarounds to get ***MySQL*** to work! The simplest is to use the binary version of ***MySQL*** (not the RPM) for Linux x86. One nice aspect of this version is that it's probably 10% faster than any version you would compile yourself! 10.3 Comment la compilation et le link affecte la vitesse de MySQL.

One known problem with the binary distribution is that with older Linux systems that use `libc` (like RedHat 4.x or Slackware), you will get some non−fatal problems with hostname resolution 4.6.3.1 Linux notes.

`isamchk` hangs with `libc.so.5.3.12`. Upgrading to the newest `libc` fixes this problem.

When using LinuxThreads you will see a minimum of three processes running. These are in fact threads. There will be one thread for the LinuxThreads manager, one thread to handle connections, and one thread to handle alarms and signals.

If you see a dead `mysqld` daemon process with `ps`, this usually means that you have found a bug in ***MySQL*** or you have got a corrupted table. 18.1 Que faire si MySQL plante constamment?.

If you are using LinuxThreads and `mysqladmin shutdown` doesn't work, you must upgrade to LinuxThreads 0.7.1 or newer.

If you are using RedHat, you might get errors like this:

```
/usr/bin/perl is needed...
/usr/sh is needed...
```

```
/usr/sh is needed...
```

If so, you should upgrade your version of `rpm` to `` `rpm−2.4.11−1.i386.rpm' `` and `` `rpm−devel−2.4.11−1.i386.rpm' `` (or later).

You can get the upgrades of libraries to RedHat 4.2 from ftp://ftp.redhat.com/updates/4.2/i386. Or http://www.sunsite.unc.edu/pub/Linux/distributions/redhat/code/rpm/ for other distributions.

If you are linking your own ***MySQL*** client and get the error:

```
ld.so.1: ./my: fatal: libmysqlclient.so.4: open failed: No such file or directory
```

when executing them, the problem can be avoided by one of the following méthodes:

- Link the client with the following flag (instead of `−Lpath`): `−Wl,r/path−libmysqlclient.so`.
- Copy `libmysqclient.so` to `` `/usr/lib' ``.
- Add the pathname of the directory where `libmysqlclient.so` is located to the `LD_RUN_PATH` environment variable before running your client.

If you are using the Fujitsu compiler `(fcc / FCC)` you will have some problems compiling ***MySQL*** because the Linux header files are very `gcc` oriented.

The following `configure` line should work with `fcc/FCC`:

```
CC=fcc CFLAGS="−O −K fast −K lib −K omitfp −Kpreex −D_GNU_SOURCE −DCONST=const
−DNO_STRTOLL_PROTO" CXX=FCC CXXFLAGS="−O −K fast −K lib  −K omitfp −K preex
−−no_exceptions −−no_rtti −D_GNU_SOURCE −DCONST=const −Dalloca=__builtin_alloca
−DNO_STRTOLL_PROTO '−D_EXTERN_INLINE=static __inline'" ./configure
−−prefix=/usr/local/mysql −−enable−assembler −−with−mysqld−ldflags=−all−static
−−disable−shared −−with−low−memory
```

4.11.5.1 Linux−x86

MySQL requires `libc` version 5.4.12 or newer. It's known to work with `libc` 5.4.46. `glibc` version 2.0.6 and later should also work. There have been some problems with the `glibc` RPMs from RedHat so if you have problems, check whether or not there are any updates! The `glibc` 2.0.7−19 and 2.0.7−29 RPMs are known to work.

On some older Linux distributions, `configure` may produce an error like this:

```
Syntax error in sched.h. Change _P to __P in the /usr/include/sched.h file.
See the Installation chapter in the Reference Manual.
```

Just do what the error message says and add an extra underscore to the `_P` macro that has only one underscore, then try again.

You may get some warnings when compiling; those shown below can be ignored:

```
mysqld.cc −o objs−thread/mysqld.o
mysqld.cc: In function `void init_signals()':
mysqld.cc:315: warning: assignment of negative value `−1' to `long unsigned int'
mysqld.cc: In function `void * signal_hand(void *)':
mysqld.cc:346: warning: assignment of negative value `−1' to `long unsigned int'
```

In Debian GNU/Linux, if you want ***MySQL*** to start automatically when the system boots, do the following:

```
shell62; cp support−files/mysql.server /etc/init.d/mysql.server
shell62; /usr/sbin/update−rc.d mysql.server defaults 99
```

`mysql.server` can be found in the `` `share/mysql' `` directory under the ***MySQL*** installation directory, or in the `` `support−files' `` directory of the ***MySQL*** source tree.

If `mysqld` always core dumps when it starts up, the problem may be that you have an old `` `/lib/libc.a' ``. Try renaming it, then remove `` `sql/mysqld' `` and do a new `make install` and try again. This problem has been reported on some Slackware installations. RedHat 5.0 has also a similar problem with some new `glibc` versions. 4.11.5.2 RedHat 5.0.

If you get the following error when linking `mysqld`, it means that your `` `libg++.a' `` is not installed correctly:

```
/usr/lib/libc.a(putc.o): In function `_IO_putc':
putc.o(.text+0x0): multiple definition of `_IO_putc'
```

You can avoid using `` `libg++.a' `` by running `configure` like this:

```
shell62; CXX=gcc ./configure
```

4.11.5.2 RedHat 5.0

If you have any problems with ***MySQL*** on RedHat, you should start by upgrading `glibc` to the newest possible version!

If you install all the official RedHat patches (including `glibc−2.0.7−19` and `glibc−devel−2.0.7−19`), both the binary and source distributions of ***MySQL*** should work without any trouble!

The updates are needed since there is a bug in `glibc` 2.0.5 in how `pthread_key_create` variables are freed. With `glibc` 2.0.5, you must use a statically−linked ***MySQL*** binary distribution. If you want to compile from source, you must install the corrected version of LinuxThreads from http://www.mysql.com/Downloads/Linux or upgrade your `glibc`.

If you have an incorrect version of `glibc` or LinuxThreads, the symptom is that `mysqld` crashes after each connection. For example, `mysqladmin version` will crash `mysqld` when it finishes!

Another symptom of incorrect libraries is that `mysqld` crashes at once when it starts. On some Linux systems, this can be fixed by configuring like this:

```
shell62; ./configure −−with−mysqld−ldflags=−all−static
```

On Redhat 5.0, the easy way out is to install the `glibc` 2.0.7−19 RPM and run `configure` ***without*** the `−−with−mysqld−ldflags=−all−static` option.

For the source distribution of `glibc` 2.0.7, a patch that is easy to apply and is tested with ***MySQL*** may be found at:

http://www.mysql.com/Download/Linux/glibc−2.0.7−total−patch.tar.gz

If you experience crashes like these when you build ***MySQL***, you can always download the newest binary

version of ***MySQL***. This is statically−linked to avoid library conflicts and should work on all Linux systems!

MySQL comes with an internal debugger that can generate trace files with a lot of information that can be used to find and solve a wide range of different problems. G.1 Debugguer un serveur MySQL.

4.11.5.3 RedHat 5.1

The `glibc` of RedHat 5.1 (`glibc` 2.0.7−13) has a memory leak, so to get a stable ***MySQL*** version, you must upgrade `glibc` to 2.0.7−19, downgrade `glibc` or use a binary version of `mysqld`. If you don't do this, you will encounter memory problems (out of memory, etc., etc.). The most common error in this case is:

```
Can't create a new thread (errno 11). If you are not out of available
memory, you can consult the manual for any possible OS dependent bug
```

After you have upgraded to `glibc` 2.0.7−19, you can configure ***MySQL*** with dynamic linking (the default), but you ***cannot*** run `configure` with the `--with−mysqld−ldflags=−all−static` option until you have installed `glibc` 2.0.7−19 from source!

You can check which version of `glibc` you have with `rpm −q glibc`.

4.11.5.4 Linux−SPARC

In some implementations, `readdir_r()` is broken. The symptom is that `SHOW DATABASES` always returns an empty set. This can be fixed by removing `HAVE_READDIR_R` from `` `config.h' `` after configuring and before compiling.

Some problems will require patching your Linux installation. The patch can be found at http://www.mysql.com/patches/Linux−sparc−2.0.30.diff. This patch is against the Linux distribution `` `sparclinux−2.0.30.tar.gz' `` that is available at `vger.rutgers.edu` (a version of Linux that was never merged with the official 2.0.30). You must also install LinuxThreads 0.6 or newer.

Thanks to jacques@solucorp.qc.ca for this information.

4.11.5.5 Linux−Alpha

The big problem on Linux−Alpha is that there are still some problems with threads in `glibc` on this platform. You should start by getting the newest `glibc` version you can find.

Note that before you run any programs that use threads (like `mysqld`, `thr_alarm` or `thr_lock`), you should raise the shared memory limit (with `ulimit`). The ***MySQL*** benchmarks are known to fail if you forget to do this!

Configure ***MySQL*** with the following command:

```
shell62; CC=gcc CCFLAGS="−Dalpha_linux_port" \
       CXX=gcc CXXFLAGS="−O3 −Dalpha_linux_port −felide−constructors −fno−exceptions −fno−rtti" \
       ./configure −−prefix=/usr/local/mysql
```

Try to compile `mysys/thr_lock` and `mysys/thr_alarm`. Test that these programs work! (Invoke each one with no arguments. Each should end with `test_succeeded` if everything was okay.)

After installing ***MySQL***, uncomment the `ulimit` command in `safe_mysqld` and add options to increase

shared memory.

Note that Linux−Alpha is still an alpha−quality platform for ***MySQL***. With the newest `glibc`, you have a very good chance of it working.

If you have problems with signals (***MySQL*** dies unexpectedly under high load) you may have found an OS bug with threads and signals. In this case you can tell ***MySQL*** not to use signals by configuring with:

```
shell62; CFLAGS=−DDONT_USE_THR_ALARM \
       CXXFLAGS=−DDONT_USE_THR_ALARM \
       ./configure ...
```

This doesn't affect the performance of ***MySQL***, but has the side effect that you can't kill clients that are ``sleeping'' on a connection with `mysqladmin kill` or `mysqladmin shutdown`. Instead, the client will die when it issues its next command.

4.11.5.6 MkLinux

MySQL should work on MkLinux with the newest `glibc` package (tested with `glibc` 2.0.7).

4.11.5.7 Qube2 Linux

To get ***MySQL*** to work on Qube2, (Linux Mips), you need the newest `glibc` libraries (`glibc−2.0.7−29C2` is known to work). You must also use the `egcs` C++ compiler (`egcs−1.0.2−9` or newer).

4.11.6 Alpha−DEC−Unix

When compiling threaded programs under Digital UNIX, the documentation recommends using the `−pthread` option for `cc` and `cxx` and the libraries `−lmach −lexc` (in addition to `−lpthread`). You should run `configure` something like this:

```
shell62; CC="cc −pthread" CXX="cxx −pthread −O" \
       ./configure −−with−named−thread−libs="−lpthread −lmach −lexc −lc"
```

When compiling `mysqld`, you may see a couple of warnings like this:

```
mysqld.cc: In function void handle_connections()':
mysqld.cc:626: passing long unsigned int *' as argument 3 of
accept(int,sockadddr *, int *)'
```

You can safely ignore these warnings. They occur because `configure` can detect only errors, not warnings.

If you start the server directly from the command line, you may have problems with it dying when you log out. (When you log out, your outstanding processes receive a `SIGHUP` signal.) If so, try starting the server like this:

```
shell62; nohup mysqld [options] 38;
```

`nohup` causes the command following it to ignore any `SIGHUP` signal sent from the terminal. Alternatively, start the server by running `safe_mysqld`, which invokes `mysqld` using `nohup` for you.

4.11.7 Alpha−DEC−OSF1

If you have problems compiling and have DEC `CC` and `gcc` installed, try running `configure` like this:

```
shell62; CC=cc CFLAGS=−O CXX=gcc CXXFLAGS=−O3 \
       ./configure −−prefix=/usr/local/mysql
```

If you get problems with the `` `c_asm.h' `` file, you can create and use a 'dummy' `` `c_asm.h' `` file with:

```
shell62; touch include/c_asm.h
shell62; CC=gcc CFLAGS=−I./include \
       CXX=gcc CXXFLAGS=−O3 \
       ./configure −−prefix=/usr/local/mysql
```

On OSF1 V4.0D and compiler "DEC C V5.6−071 on Digital UNIX V4.0 (Rev. 878)" the compiler had some strange behavior (undefined `asm` symbols). `/bin/ld` also appears to be broken (problems with `_exit undefined` errors occuring while linking `mysqld`). On this system, we have managed to compile ***MySQL*** with the following `configure` line, after replacing `/bin/ld` with the version from OSF 4.0C:

```
shell62; CC=gcc CXX=gcc CXXFLAGS=−O3 ./configure −−prefix=/usr/local/mysql
```

In some versions of OSF1, the `alloca()` function is broken. Fix this by removing the line in `` `config.h' `` that defines `'HAVE_ALLOCA'`.

The `alloca()` function also may have an incorrect prototype in `/usr/include/alloca.h`. This warning resulting from this can be ignored.

`configure` will use the following thread libraries automatically:
`−−with−named−thread−libs="−lpthread −lmach −lexc −lc"`.

When using `gcc`, you can also try running `configure` like this:

```
shell62; CFLAGS=−D_PTHREAD_USE_D4 CXX=gcc CXXFLAGS=−O3 ./configure ....
```

If you have problems with signals (***MySQL*** dies unexpectedly under high load) you may have found an OS bug with threads and signals. In this case you can tell ***MySQL*** not to use signals by configuring with:

```
shell62; CFLAGS=−DDONT_USE_THR_ALARM \
       CXXFLAGS=−DDONT_USE_THR_ALARM \
       ./configure ...
```

This doesn't affect the performance of ***MySQL***, but has the side effect that you can't kill clients that are ``sleeping'' on a connection with `mysqladmin kill` or `mysqladmin shutdown`. Instead, the client will die when it issues its next command.

4.11.8 SGI−Irix

You may have to undefine some things in `` `config.h' `` after running `configure` and before compiling.

In some Irix implementations, the `alloca()` function is broken. If the `mysqld` server dies on some `SELECT` statements, remove the lines from `` `config.h' `` that define `HAVE_ALLOC` and `HAVE_ALLOCA_H`. If `mysqladmin create` doesn't work, remove the line from `` `config.h' `` that

defines HAVE_READDIR_R. You may have to remove the HAVE_TERM_H line as well.

SGI recommends that you install all of the patches on this page as a set:
http://support.sgi.com/surfzone/patches/patchset/6.2_indigo.rps.html

At the very minimum, you should install the latest kernel rollup, the latest `rld` rollup, and the latest `libc` rollup.

You definately need all the POSIX patches on this page, for pthreads support:

http://support.sgi.com/surfzone/patches/patchset/6.2_posix.rps.html

If you get the something like the following error when compiling `` `mysql.cc' ``:

```
"/usr/include/curses.h", line 82: error(1084): invalid combination of type
```

Then type the following in the top−level directory of your ***MySQL*** source tree:

```
shell62; extra/replace bool curses_bool 60; /usr/include/curses.h 62; include/curses.h
shell62; make
```

There have also been reports of scheduling problems. If only one thread is running, things go slow. Avoid this by starting another client. This may lead to a 2−to−10−fold increase in execution speed thereafter for the other thread. This is a poorly−understood problem with Irix threads; you may have to improvise to find solutions until this can be fixed.

If you are compiling with `gcc`, you can use the following `configure` command:

```
shell62; CC=gcc CXX=gcc CXXFLAGS=−O3 \
       ./configure −−prefix=/usr/local/mysql −−with−thread−safe−client \
       −−with−named−thread−libs=−lpthread
```

4.11.9 FreeBSD

If you notice that `configure` will use MIT−pthreads, you should read the MIT−pthreads notes. 4.9 Remarques sur MIT−pthreads.

If you get an error from `make install` that it can't find `` `/usr/include/pthreads' ``, `configure` didn't detect that you need MIT−pthreads. This is fixed by executing these commands:

```
shell62; rm config.cache
shell62; ./configure −−with−mit−threads
```

The behavior of FreeBSD `make` is slightly different from that of GNU `make`. If you have `make`−related problems, you should install GNU `make`.

If `mysql` or `mysqladmin` takes a long time to respond, a user said the following:

Are you running the `ppp` user process? On one FreeBSD box (2.2.5) ***MySQL*** clients takes a couple of seconds to connect to `mysqld` if the `ppp` process is running.

FreeBSD is also known to have a very low default file handle limit. 18.10 File not found.

If you have a problem with `SELECT NOW()` returning values in GMT and not your local time, you have to set the `TZ` environment variable to your current timezone. This should be done for the environment in which the server runs, for example, in `safe_mysqld` or `mysql.server`.

Make sure that the `localhost` entry in the `` `/etc/hosts' `` file is correct (otherwise you will have problems connecting to the database). The `` `/etc/hosts' `` file should start with a line:

```
127.0.0.1       localhost localhost.your.domain
```

If you are using FreeBSD 2.2.6, don't forget to apply the ttcp and mmap−22 patches to the OS (for security reasons). Please see http://www.freebsd.org for these CERT patches.

If you are using FreeBSD 2.2.7 and you have problems killing the `mysqld` daemon, you should get new sources using `cvsup` and recompile `libc_r`.

4.11.9.1 FreeBSD−3.0

You have to run `configure` with the `−−with−named−thread−libs=−lc_r` option.

The pthreads library for FreeBSD doesn't contain the `sigwait()` function and there are some bugs in it. To fix this, get the `` `FreeBSD−3.0−libc_r−1.0.diff' `` file from the FreeBSD area of the ***MySQL*** FTP site and apply it in the `` `/usr/src/lib/libc_r/uthread' `` directory. Then follow the instructions that can be found with `man pthread` about how to recompile the `libc_r` library.

You can test if you have a ``modern'' `libpthread.a` with this command:

```
shell62; nm /usr/lib/libc_r.a | grep sigwait
```

If the above doesn't find `sigwait`, you must use the patch above and recompile `libc_r`.

4.11.10 NetBSD

To compile on NetBSD you need GNU `make`. Otherwise the compile will crash when `make` tries to run `lint` on C++ files.

4.11.11 BSD/OS

4.11.11.1 BSD/OS 2.x

If you get the following error when compiling ***MySQL***, your `ulimit` value for virtual memory is too low:

```
item_func.h: In méthode `Item_func_ge::Item_func_ge(const Item_func_ge 38;)':
item_func.h:28: virtual memory exhausted
make[2]: *** [item_func.o] Error 1
```

Try using `ulimit −v 80000` and run `make` again. If this doesn't work and you are using `bash`, try switching to `csh` or `sh`; some BSDI users have reported problems with `bash` and `ulimit`.

If you are using `gcc`, you may also use have to use the `−−with−low−memory` flag for `configure` to be able to compile `` `sql_yacc.cc' ``.

If you have a problem with `SELECT NOW()` returning values in GMT and not your local time, you have to set the `TZ` environment variable to your current timezone. This should be done for the environment in which the server runs, for example in `safe_mysqld` or `mysql.server`.

4.11.11.2 BSD/OS 3.x

Upgrade to BSD/OS 3.1. If that is not possible, install BSDIpatch M300−038.

Use the following command when configuring ***MySQL***:

```
shell62; env CXX=shlicc++ CC=shlicc2 \
       ./configure \
           --prefix=/usr/local/mysql \
           --localstatedir=/var/mysql \
           --without-perl \
           --with-unix-socket-path=/var/mysql/mysql.sock
```

The following is also known to work:

```
shell62; env CC=gcc CXX=gcc CXXFLAGS=-O3 \
       ./configure \
           --prefix=/usr/local/mysql \
           --with-unix-socket-path=/var/mysql/mysql.sock
```

You can change the directory locations if you wish, or just use the defaults by not specifying any locations.

If you have problems with performance under heavy load, try using the `--skip-thread-priority` option to `safe_mysqld`! This will run all threads with the same priority; on BSDI 3.1, this gives better performance (at least until BSDI fixes their thread scheduler).

If you get the error `virtual memory exhausted` while compiling, you should try using `ulimit -v 80000` and run `make` again. If this doesn't work and you are using `bash`, try switching to `csh` or `sh`; some BSDI users have reported problems with `bash` and `ulimit`.

4.11.11.3 BSD/OS 4.x

BSDI 4.x has some thread related bugs. If you want to use ***MySQL*** on this, you should install all thread related patches. At least M400−023 should be installed.

4.11.12 SCO

The current port is tested only on a ``sco3.2v5.0.4" and ``sco3.2v5.0.5" system. There has also been a lot of progress on a port to ``sco 3.2v4.2".

1. For OpenServer 5.0.X you need to use GDS in Skunkware 95 (95q4c). This is necessary because GNU `gcc` 2.7.2 in Skunkware 97 does not have GNU `as`. You can also use `egcs` 1.1.2 or newer http://www.egcs.com/. If you are using `egcs` 1.1.2 you have to execute the following command:
   ```
   shell62; cp -p /usr/include/pthread/stdtypes.h
   /usr/local/lib/gcc-lib/i386-pc-sco3.2v5.0.5/egcs-2.91.66/include/pthread/
   ```
2. You need the port of GCC 2.5.? for this product and the Development system. They are required on this version of SCO UNIX. You cannot just use the GCC Dev system.
3. You should get the FSU Pthreads package and install it first. This can be found at http://www.cs.wustl.edu/~schmidt/ACE_wrappers/FSU−threads.tar.gz. You can also get a precompiled package from ftp://www.mysql.com/pub/mysql/Downloads/SCO/FSU−threads−3.5c.tar.gz.

4. FSU Pthreads can be compiled with SCO UNIX 4.2 with tcpip. Or OpenServer 3.0 or Open Desktop 3.0 (OS 3.0 ODT 3.0), with the SCO Development System installed using a good port of GCC 2.5.X ODT or OS 3.0 you will need a good port of GCC 2.5.? There are a lot of problems without a good port. The port for this product requires the SCO UNIX Development system. Without it, you are missing the libraries and the linker that is needed.
5. To build FSU Pthreads on your system, do the following:
    1. Run `./configure` in the `` `threads/src' `` directory and select the SCO OpenServer option. This command copies `` `Makefile.SCO5' `` to `` `Makefile' ``.
    2. Run `make`.
    3. To install in the default `` `/usr/include' `` directory, login as root, then `cd` to the `` `thread/src' `` directory, and run `make install`.
6. Remember to use GNU `make` when making ***MySQL***.
7. On OSR 5.0.5, you should use the following configure line:
```
shell62; CC="gcc −DSCO" CXX="gcc −DSCO" ./configure
```
   The `−DSCO` is needed to help configure detect some thread functions properly. If you forget `−DSCO`, you will get the following error message while compiling:
```
my_pthread.c: In function `my_pthread_mutex_init':
my_pthread.c:374: `pthread_mutexattr_default' undeclared (first use this function)
```
8. If you don't start `safe_mysqld` as root, you probably will get only the default 110 open files per process. `mysqld` will write a note about this in the log file.
9. With SCO 3.2V5.0.5, you should use a FSU Pthreads version 3.5c or newer. The following `configure` command should work:
```
shell62; CC="gcc −belf" ./configure −−prefix=/usr/local/mysql −−disable−shared
```
10. With SCO 3.2V4.2, you should use a FSU Pthreads version 3.5c or newer. The following `configure` command should work:
```
shell62; CFLAGS="−D_XOPEN_XPG4" CXX=gcc CXXFLAGS="−D_XOPEN_XPG4" \
       ./configure \
           −−with−debug −−prefix=/usr/local/mysql \
           −−with−named−thread−libs="−lgthreads −lsocket −lgen −lgthreads" \
           −−with−named−curses−libs="−lcurses"
```

You may get some problems with some include files. In this case, you can find new SCO−specific include files at ftp://www.mysql.com/pub/mysql/Downloads/SCO/SCO−3.2v4.2−includes.tar.gz. You should unpack this file in the `` `include' `` directory of your ***MySQL*** source tree.

SCO development notes:

- ***MySQL*** should automatically detect FSU Pthreads and link `mysqld` with `−lgthreads −lsocket −lgthreads`.
- The SCO development libraries are reentrant in FSU Pthreads. SCO claims that its libraries' functions are reentrant, so they must be reentrant with FSU Pthreads. FSU Pthreads on OpenServer tries to use the SCO scheme to make reentrant library.
- FSU Pthreads (at least the version at `www.mysql.com`) comes linked with GNU `malloc`. If you encounter problems with memory usage, make sure that `` `gmalloc.o' `` is included in `` `libgthreads.a' `` and `` `libgthreads.so' ``.
- In FSU Pthreads, the following system calls are pthreads−aware: `read()`, `write()`, `getmsg()`, `connect()`, `accept()`, `select()` and `wait()`.

If you want to install DBI on SCO, you have to edit the `` `Makefiles' `` in DBI−xxx and each subdirectory:

```
OLD:                                    NEW:
CC = cc                                 CC = gcc −belf
CCCDLFLAGS = −KPIC −W1,−Bexport         CCCDLFLAGS = −fpic
CCDLFLAGS = −wl,−Bexport                CCDLFLAGS =

LD = ld                                 LD = gcc −belf −G −fpic
LDDLFLAGS = −G −L/usr/local/lib         LDDLFLAGS = −L/usr/local/lib
LDFLAGS = −belf −L/usr/local/lib        LDFLAGS = −L/usr/local/lib

LD = ld                                 LD = gcc −belf −G −fpic
OPTIMISE = −Od                          OPTIMISE = −O1

OLD:
CCCFLAGS = −belf −dy −w0 −U M_XENIX −DPERL_SCO5 −I/usr/local/include
```

```
NEW:
CCFLAGS = −U M_XENIX −DPERL_SCO5 −I/usr/local/include
```

This is because the Perl dynaloader will not load the `DBI` modules if they were compiled with `icc` or `cc`.

Perl works best when compiled with `cc`.

4.11.13 SCO Unixware 7.0 notes

You must use a version of ***MySQL*** at least as recent as 3.22.13, since that version fixes some portability problems under Unixware.

We have been able to compile ***MySQL*** with the following `configure` command on UnixWare 7.0.1:

```
shell62; CC=cc CXX=CC ./configure −−prefix=/usr/local/mysql
```

4.11.14 IBM−AIX

Automatic detection of `xlC` is missing from Autoconf, so a `configure` command something like this is needed when using the IBM compiler:

```
shell62; CC="xlc_r −ma −O3 −qstrict −DHAVE_INT_8_16_32" \
       CXX="xlC_r −ma −O3 −qstrict −DHAVE_INT_8_16_32" \
       ./configure
```

If you are using `egcs` to compile ***MySQL***, you ***MUST*** use the `−fno−exceptions` flag, as the exception handling in `egcs` is not thread−safe! (This is tested with `egcs` 1.1.) We recommend the following `configure` line with `egcs` and `gcc` on AIX:

```
shell62; CXX=gcc \
       CXXFLAGS="−felide−constructors −fno−exceptions −fno−rtti" \
       ./configure −−prefix=/home/monty −−with−debug −−with−low−memory
```

If you have problems with signals (***MySQL*** dies unexpectedly under high load) you may have found an OS bug with threads and signals. In this case you can tell ***MySQL*** not to use signals by configuring with:

```
shell62; CFLAGS=−DDONT_USE_THR_ALARM CXX=gcc \
       CXXFLAGS="−felide−constructors −fno−exceptions −fno−rtti −DDONT_USE_THR_ALARM" \
       ./configure −−prefix=/home/monty −−with−debug −−with−low−memory
```

This doesn't affect the performance of ***MySQL***, but has the side effect that you can't kill clients that are ``sleeping'' on a connection with `mysqladmin kill` or `mysqladmin shutdown`. Instead, the client will die when it issues its next command.

4.11.15 HP−UX

There are a couple of ``small'' problems when compiling ***MySQL*** on HP−UX. We recommend that you use `gcc` instead of the HP−UX native compiler, because `gcc` produces better code!

We recommend one to use gcc 2.95 on HP−UX. Don't use high optimization flags (like −O6) as this may not be safe on HP−UX.

Note that MIT−pthreads can't be compiled with the HP−UX compiler, because it can't compile `.S` (assembler) files.

The following configure line should work:

```
CFLAGS="−DHPUX −I/opt/dce/include" CXXFLAGS="−DHPUX −I/opt/dce/include
−felide−constructors −fno−exceptions −fno−rtti" CXX=gcc ./configure −−with−pthread
−−with−named−thread−libs='−ldce' −−prefix=/usr/local/mysql −−disable−shared
```

If you are compiling `gcc` 2.95 yourself, you should NOT link it with the DCE libraries (`libdce.a` or `libcma.a`) if you want to compile ***MySQL*** with MIT−pthreads. If you mix the DCE and MIT−pthreads packages you will get a `mysqld` to which you cannot connect. Remove the DCE libraries while you compile `gcc` 2.95!

4.12 Remarques pour Win32

This section describes installation and use of ***MySQL*** on Win32. This is also described in the `` `README' `` file that comes with the ***MySQL*** Win32 distribution.

4.12.1 Installer strong{MySQL} sous Win32

If you don't have a registered version of ***MySQL***, you should first download the shareware version from:

MySQL 3.21.29

If you plan to connect to ***MySQL*** from some other program, you will probably also need the ***MyODBC*** driver. You can find this at the ***MySQL*** download page.

To install either distribution, unzip it in some empty directory and run the `Setup.exe` program.

Installation takes place in `` `C:\mysql' ``.

4.12.2 Démarrer *MySQL* sous Win95 / Win98

MySQL uses TCP/IP to connect a client to a server. (This will allow any machine on your network to connect to your ***MySQL*** server). Because of this, you must install TCP/IP on your machine before starting ***MySQL***. You can find TCP/IP on your Windows CD−ROM.

Note that if you are using an old Win95 release (for example OSR2), it's likely that you have an old Winsock package! ***MySQL*** requires Winsock 2! You can get the newest Winsock from Microsoft. Win98 has as default the new Winsock 2 library, so the above doesn't apply for Win98.

There are 2 different ***MySQL*** servers you can use:

`mysqld`	Compiled with full debugging and automatic memory allocation checking
`mysqld−opt`	Optimized for a Pentium processor.

Both of the above should work on any Intel processor >= i386.

To start the `mysqld` server, you should start a MS−DOS window and type:

```
C:\mysql\bin\mysqld
```

This will start `mysqld` in the background without a window.

You can kill the ***MySQL*** server by executing:

```
C:\mysql\bin\mysqladmin −u root shutdown
```

Note that Win95/Win98 don't support creation of named pipes. On Win95/Win98, you can only use named pipes to connect to a remote ***MySQL*** running on an NT server.

4.12.3 Démarrer *MySQL* sous NT

The Win95/Win98 section also applies to ***MySQL*** on NT, with the following differences:

To get ***MySQL*** to work with TCP/IP, you must install service pack 3 (or newer)!

For NT, the server name is `mysqld−nt`. Normally you should install ***MySQL*** as a service on NT:

```
C:\mysql\bin\mysqld−nt −−install
```

(You could use the `mysqld` or `mysqld−opt` servers on NT, but those cannot be started as a service or use named pipes.)

You can start and stop the ***MySQL*** service with:

```
NET START mysql
NET STOP mysql
```

Note that in this case you can't use any other options for `mysqld−nt`!

You can also run `mysqld−nt` as a standalone program on NT if you need to start `mysqld−nt` with any options! If you start `mysqld−nt` without options on NT, `mysqld−nt` tries to starts itself as a service with the default service options. If you have stopped `mysqld−nt`, you have to start it with `NET START mysql`.

The service is installed with the name `MySql`. Once installed, it must be started using Services Control Manager (SCM) Utility (found in Control Panel) or by using the `NET START MySQL` command. If any options are desired, they must be specified as "Startup parameters" in the SCM utility before you start the ***MySQL*** service. Once running, `mysqld−nt` can be stopped using `mysqladmin` or from the SCM utility or by using the command `NET STOP MySQL`. If you use SCM to stop `mysqld−nt`, there is a strange message from SCM about `mysqld shutdown normally`. When run as a service, `mysqld−nt` has no access to a console and so no messages can be seen.

On NT you can get the following service error messages:

Permission Denied	Means that it cannot find `mysqld−nt.exe`
Cannot Register	Means that the path is incorrect

If you have problems installing `mysqld−nt` as a service, try starting it with the full path:

```
C:\mysql\bin\mysqld --install
```

If this doesn't work, you can get `mysqld-nt` to start properly by fixing the path in the registry!

If you don't want to start `mysqld-nt` as a service, you can start it as follows:

```
C:\mysql\bin\mysqld-nt --standalone
```

or

```
C:\mysql\bin\mysqld-nt --standalone --debug
```

The last version gives you a debug trace in `` `C:\mysqld.trace' ``.

4.12.4 Faire tourner *MySQL* sous Win32

MySQL supports TCP/IP on all Win32 platforms and named pipes on NT. The default is to use named pipes for local connections on NT and TCP/IP for all other cases if the client has TCP/IP installed. The host name specifies which protocol is used:

protocolNULL (none) On NT, try named pipes first; if that doesn't work, use TCP/IP. On Win95/Win98, TCP/IP is used. . Named pipes localhost TCP/IP to current host hostname TCP/IP

You can force a ***MySQL*** client to use named pipes by specifying the `--pipe` option. Use the `--socket` option to specify the name of the pipe.

You can test whether or not ***MySQL*** is working by executing the following commands:

```
C:\mysql\bin\mysqlshow
C:\mysql\bin\mysqlshow -u root mysql
C:\mysql\bin\mysqladmin version status proc
C:\mysql\bin\mysql test
```

By default, ***MySQL***−Win32 is configured to be installed in `` `C:\mysql' ``. If you want to install ***MySQL*** elsewhere, install it in `` `C:\mysql' ``, then move the installation to where you want it. If you do move ***MySQL***, you must tell `mysqld` where everything is by supplying options to `mysqld`. Use `C:\mysql\bin\mysqld --help` to display all options! If, for example, you have moved the ***MySQL*** distribution to `` `D:\programs\mysql' ``, you must start `mysqld` with:
`D:\programs\mysql\bin\mysqld --basedir D:\programs\mysql`

With the registered version of ***MySQL***, you can also create a `` `C:\my.cnf' `` file that holds any default options for the ***MySQL*** server. Copy the file `` `\mysql\my-example.cnf' `` to `` `C:\my.cnf' `` and edit this to suit your setup. Note that you should specify all paths with `/` instead of `\`. If you use `\`, you need to specify this twice, as `\` is the escape character in ***MySQL***. 4.15.4 Fichier d'options.

If `mysqld` is slow to answer to connections on Win95/Win98, there is probably a problem with your DNS. In this case, start `mysqld` with `--skip-name-resolve` and use only `localhost` and IP numbers in the ***MySQL*** grant tables. You can also avoid DNS when connecting to a `mysqld-nt` ***MySQL*** server running on NT by using the `--pipe` argument to specify use of named pipes. This works for most ***MySQL*** clients.

There are two versions of the ***MySQL*** command line tool:

`mysql`	Compiled on native Win32, which offers very limited text editing capabilities.
`mysqlc`	Compiled with the Cygnus GNU compiler and libraries, which offers `readline` editing.

If you want to use `mysqlc.exe`, you must copy `` `C:\mysql\lib\cygwinb19.dll' `` to `` `\windows\system' `` (or similar place).

The default privileges on Win32 give all local users full privileges to all databases. To make ***MySQL*** more secure, you should set a password for all users and remove the row in the `mysql.user` table that has `Host='localhost'` and `User=''`.

You should also add a password for the `root` user:

```
C:\mysql\bin\mysql mysql
mysql62; DELETE FROM user WHERE Host='localhost' AND User='';
mysql62; QUIT
C:\mysql\bin\mysqladmin reload
C:\mysql\bin\mysqladmin −u root password your_password
```

After you've set the password, if you want to take down the `mysqld` server, you can do so using this command:

```
mysqladmin −−user=root −−password=your_password shutdown
```

4.12.5 Se connecter à un serveur lointain *MySQL* avec Win32 et SSH

Here is a note about how to connect to get a secure connection to remote MySQL server with SSH (by David Carlson).

- Install SSH client on your windows machine − I used a free SSH client from http://www.doc.ic.ac.uk/~ci2/ssh/. Other useful links: http://www.npaci.edu/Security/npaci_security_software.html and http://www.npaci.edu/Security/samples/ssh32_windows/index.html.
- Start SSH. Set Host Name = yourmysqlserver name or IP address. Set userid=your userid to log in to your server
- Click on "local forwards". Set `local port: 3306`, `host: localhost`, `remote port: 3306`
- Save everything, otherwise you'll have to redo it the next time.
- Log in to your server with SSH.
- Start some ODBC application (for example Access)
- Create a new file and link to mySQL using the ODBC driver the same way you normally do except for server, user "localhost".

That's it. It works very well with a direct Internet connection. I'm having problems with SSH conflicting with my Win95 network and Wingate − but that'll be the topic of a posting on another software company's usegroup!

4.12.6 *MySQL*−Win32 comparé à Unix *MySQL*

MySQL−Win32 has by now proven itself to be very stable. This version of ***MySQL*** has the same features as the corresponding Unix version with the following exceptions:

Win95 and threads
: Win95 leaks about 200 bytes of main memory for each thread creation. Because of this, you shouldn't run `mysqld` for an extended time on Win95 if you do many connections, since each connection in ***MySQL*** creates a new thread! WinNT and Win98 don't suffer from this bug.

Blocking read
: ***MySQL*** uses a blocking read for each connection. This means that:
  - ◊ A connection will not be disconnected automatically after 8 hours, as happens with the Unix version of ***MySQL***.
  - ◊ If a connection ``hangs,'' it's impossible to break it without killing ***MySQL***.
  - ◊ `mysqladmin kill` will not work on a sleeping connection.
  - ◊ `mysqladmin shutdown` can't abort as long as there are sleeping connections.

  We plan to fix this in the near future.

UDF functions
For the moment, ***MySQL***−Win32 does not support user definable functions.

`DROP DATABASE`
You can't drop a database that is in use by some thread.

Killing MySQL from the task manager
You can't kill ***MySQL*** from the task manager or with the shutdown utility in Windows95. You must take it down with `mysqladmin shutdown`.

Case−insensitive names
Filenames are case insensitive on Win32, so database and table names are also case insensitive in ***MySQL*** for Win32. The only restriction is that database and table names must be given in the same case throughout a given statement. The following query would not work because it refers to a table both as `my_table` and as `MY_TABLE`:

```
SELECT * FROM my_table WHERE MY_TABLE.col=1;
```

The `\' directory character
Pathname components in Win95 are separated by `\'` characters, which is also the escape character in ***MySQL***. If you are using `LOAD DATA INFILE` or `SELECT ... INTO OUTFILE`, you must double the `\'` character or use Unix style filenames `/'` characters:

```
LOAD DATA INFILE "C:\\tmp\\skr.txt" INTO TABLE skr;
SELECT * FROM skr INTO OUTFILE 'C:/tmp/skr.txt';
```

`Can't open named pipe` error
If you use the shareware version of ***MySQL***−Win32 on NT with the newest mysql−clients you will get the following error:

```
error 2017: can't open named pipe to host: . pipe...
```

This is because the release version of ***MySQL*** uses named pipes on NT by default. You can avoid this error by using the `−−host=localhost` option to the new ***MySQL*** clients or create a file `C:\my.cnf'` that contains the following information:

```
[client]
host = localhost
```

`Access denied for user` error
If you get the error `Access denied for user: 'some−user@unknown' to database 'mysql'` when accessing a ***MySQL*** server on the same machine, this means that ***MySQL*** can't resolve your host name properly. To fix this, you should create a file `\windows\hosts'` with the following information:

```
127.0.0.1       localhost
```

Here are some open issues for anyone who might want to help us with the Win32 release:

- Make a single user `MYSQL.DLL` server. This should include everything in a standard ***MySQL*** server, except thread creation. This will make ***MySQL*** much easier to use in applications that don't need a true client/server and don't need to access the server from other hosts.
- Add some nice ``start'' and ``shutdown'' icons to the ***MySQL*** installation.
- Create a tool to manage registry entries for the ***MySQL*** startup options. The registry entry reading is already coded into `mysqld.cc`, but it should be recoded to be more ``parameter'' oriented. The tool should also be able to update the `\my.cnf'` file if the user would prefer to use this instead of the registry.
- When registering `mysqld` as a service with `−−install` (on NT) it would be nice if you could also add default options on the command line. For the moment, the workaround is to update the `C:\my.cnf'` file instead.
- When you suspend a laptop running Win95, the `mysqld` daemon doesn't accept new connections when the laptop is resumed. We don't know if this is a problem with Win95, TCP/IP or ***MySQL***.
- It would be real nice to be able to kill `mysqld` from the task manager. For the moment, you must use `mysqladmin shutdown`.
- Port `readline` to Win32 for use in the `mysql` command line tool.
- GUI versions of the standard ***MySQL*** clients (`mysql`, `mysqlshow`, `mysqladmin`, and `mysqldump`) would be nice.
- It would be nice if the socket ``read'' and ``write'' functions in `net.c'` were interruptible. This would make it possible to kill open threads with `mysqladmin kill` on Win32.
- Documentation of which Windows programs work with ***MySQL***−Win32/***MyODBC*** and what must be done to get them working.
- `mysqld` always starts in the "C" locale and not in the default locale. We would like to have `mysqld` use the current locale for the sort order.
- Port `sqlclient` to Win32 (almost done) and add more features to it!
- Add more options to MysqlManager.
- Change the communication protocol between the server and client to use Windows internal communication instead of sockets and TCP/IP.
- Implement UDF functions with `.DLL`s.
- Add macros to use the faster thread−safe increment/decrement méthodes provided by Win32.

Other Win32−specific issues are described in the `README'` file that comes with the ***MySQL***−Win32 distribution.

4.13 Remarques pour OS/2

MySQL uses quite a few open files. Because of this, you should add something like the following to your `CONFIG.SYS' file:

```
SET EMXOPT=−c −n −h1024
```

If you don't do this, you will probably run into the following error:

```
File 'xxxx' not found (Errcode: 24)
```

When using ***MySQL*** with OS/2 Warp 3, FixPack 29 or above is required. With OS/2 Warp 4, FixPack 4 or above is required. This is a requirement of the Pthreads library. ***MySQL*** must be installed in a partition that supports long file names such as HPFS, FAT32, etc.

The `INSTALL.CMD' script must be run from OS/2's own `CMD.EXE' and may not work with replacement shells such as `4OS2.EXE'.

The `scripts/mysql−install−db' script has been renamed: it is now called `install.cmd' and is a REXX script which will set up the default ***MySQL*** security settings and create the WorkPlace Shell icons for ***MySQL***.

Dynamic module support is compiled in but not fully tested. Dynamic modules should be compiled using the Pthreads runtime library.

```
gcc −Zdll −Zmt −Zcrtdll=pthrdrtl −I../include −I../regex −I.. \
    −o example udf_example.cc −L../lib −lmysqlclient udf_example.def
mv example.dll example.udf
```

Note: Due to limitations in OS/2, UDF module name stems must not exceed 8 characters. Modules are stored in the `/mysql2/udf' directory; the `safe−mysqld.cmd` script will put this directory in the `BEGINLIBPATH` environment variable. When using UDF modules, specified extensions are ignored −− it is assumed to be `.udf'. For example, in Unix, the shared module might be named `example.so' and you would load a function from it like this:

```
CREATE FUNCTION metaphon RETURNS STRING SONAME "example.so";
```

Is OS/2, the module would be named `example.udf', but you would not specify the module extension:

```
CREATE FUNCTION metaphon RETURNS STRING SONAME "example";
```

4.14 TcX

As a service, TcX provides a set of binary distributions of ***MySQL*** that are compiled at TcX or at sites where customers kindly have given us access to their machines.

These distributions are generated with `scripts/make_binary_distribution` and are configured with the following compilers and options:

SunOS 4.1.4 2 sun4c with `gcc` 2.7.2.1

```
        CC=gcc CXX=gcc CXXFLAGS=−O3 ./configure −−prefix=/usr/local/mysql −−disable−shared
```

SunOS 5.5.1 sun4u with `egcs` 1.0.3a

```
        CC=gcc CFLAGS="−O6 −fomit−frame−pointer" CXX=gcc CXXFLAGS="−O6 −fomit−frame−pointer
        −felide−constructors −fno−exceptions −fno−rtti" ./configure −−prefix=/usr/local/mysql
        −−with−low−memory
```

SunOS 5.6 sun4u with `egcs` 2.90.27

```
        CC=gcc CFLAGS="−O6 −fomit−frame−pointer" CXX=gcc CXXFLAGS="−O6 −fomit−frame−pointer
        −felide−constructors −fno−exceptions −fno−rtti" ./configure −−prefix=/usr/local/mysql
        −−with−low−memory
```

SunOS 5.6 i86pc with `gcc` 2.8.1

```
        CC=gcc CXX=gcc CXXFLAGS=−O3 ./configure −−prefix=/usr/local/mysql −−with−low−memory
```

Linux 2.0.33 i386 with `pgcc` 2.90.29 (`egcs` 1.0.3a)

```
        CFLAGS="−O6 −mpentium −mstack−align−double −fomit−frame−pointer" CXX=gcc CXXFLAGS="−O6
        −mpentium −mstack−align−double −fomit−frame−pointer −felide−constructors −fno−exceptions
        −fno−rtti" ./configure −−prefix=/usr/local/mysql −−enable−assembler
        −−with−mysqld−ldflags=−all−static
```

SCO 3.2v5.0.4 i386 with `gcc` 2.7−95q4

```
        CC=gcc CXX=gcc CXXFLAGS=−O3 ./configure −−prefix=/usr/local/mysql
```

AIX 2 4 with `gcc` 2.7.2.2

```
        CC=gcc CXX=gcc CXXFLAGS=−O3 ./configure −−prefix=/usr/local/mysql
```

OSF1 V4.0 564 alpha with `gcc` 2.8.1

```
        CC=gcc CFLAGS=−O CXX=gcc CXXFLAGS=−O3 ./configure −−prefix=/usr/local/mysql −−with−low−memory
```

Irix 6.3 IP32 with `gcc` 2.8.0

```
        CC=gcc CXX=gcc CXXFLAGS=−O3 ./configure −−prefix=/usr/local/mysql
```

BSDI BSD/OS 3.1 i386 with `gcc` 2.7.2.1

```
        CC=gcc CXX=gcc CXXFLAGS=−O ./configure −−prefix=/usr/local/mysql
```

BSDI BSD/OS 2.1 i386 with `gcc` 2.7.2

```
        CC=gcc CXX=gcc CXXFLAGS=−O3 ./configure −−prefix=/usr/local/mysql
```

Anyone who has more optimal options for any of the configurations listed above can always mail them to the developer's mailing list at

RPM distributions prior to ***MySQL*** 3.22 are user−contributed. Beginning with 3.22, some RPMs are TcX−generated.

4.15 Paramètrage post−installation et tests

Once you've installed ***MySQL*** (from either a binary or source distribution), you need to initialize the grant tables, start the server and make sure that the server works okay. You may also wish to arrange for the server to be started and stopped automatically when your system starts up and shuts down.

Normally you install the grant tables and start the server like this:

```
shell62; cd mysql_installation_directory
shell62; ./bin/mysql_install_db
shell62; ./bin/safe_mysqld 38;
```

Testing is most easily done from the top−level directory of the ***MySQL*** distribution. For a binary distribution, this is your installation directory (typically something like `/usr/local/mysql'). For a source distribution, this is the main directory of your ***MySQL*** source tree.

In the commands shown below in this section and in the following subsections, `BINDIR` is the path to the location in which programs like `mysqladmin` and `safe_mysqld` are installed. For a binary distribution, this is the `bin' directory within the distribution. For a source distribution, `BINDIR` is probably `/usr/local/bin', unless you specified an installation directory other than `/usr/local' when you ran `configure`. `EXECDIR` is the location in which the `mysqld` server is installed. For a binary distribution, this is the same as `BINDIR`. For a source distribution, `EXECDIR` is probably `/usr/local/libexec'.

Testing is described in detail below:

1. If necessary, start the `mysqld` server and set up the initial ***MySQL*** grant tables containing the privileges that determine how users are allowed to connect to the server. This is normally done with the `mysql_install_db` script:
```
shell62; scripts/mysql_install_db
```

   Typically, `mysql_install_db` needs to be run only the first time you install ***MySQL***. Therefore, if you are upgrading an existing installation, you can skip this step. (However, `mysql_install_db` is quite safe to use and will not update any tables that already exist, so if you are unsure what to do, you can always run `mysql_install_db`.) `mysql_install_db` creates six tables (`user`, `db`, `host`, `tables_priv`, `columns_priv` and `func`) in the `mysql` database. A description of the initial privileges is given in 6.10 Droits initiaux. Briefly, these privileges allow the ***MySQL*** `root` user to do anything, and allow anybody to create or use databases with a name of `'test'` or starting with `'test_'`. If you don't set up the grant tables, the following error will appear in the log file when you start the server:

```
mysqld: Can't find file: 'host.frm'
```

   The above may also happens with a binary ***MySQL*** distribution if you don't start ***MySQL*** by executing exactly `./bin/safe_mysqld`! You might need to run `mysql_install_db` as `root`. However, if you prefer, you can run the ***MySQL*** server as an unprivileged (non−`root`) user, provided that user can read and write files in the database directory. Instructions for running ***MySQL*** as an unprivileged user are given in Changing ***MySQL*** user. If you have problems with `mysql_install_db`, see `mysql_install_db`. There are some alternatives to running the `mysql_install_db` script as it is provided in the ***MySQL*** distribution:

   - You may want to edit `mysql_install_db` before running it, to change the initial privileges that are installed into the grant tables. This is useful if you want to install ***MySQL*** on a lot of machines with the same privileges. In this case you probably should need only to add a few extra `INSERT` statements to the `mysql.user` and `mysql.db` tables!
   - If you want to change things in the grant tables after installing them, you can run `mysql_install_db`, then use `mysql −u root mysql` to connect to the grant tables as the ***MySQL*** `root` user and issue SQL statements to modify the grant tables directly.
   - It is possible to recreate the grant tables completely after they have already been created. You might want to do this if you've already installed the tables but then want to recreate them after editing `mysql_install_db`.

   For more information about these alternatives, see 6.10 Droits initiaux.
2. Start the ***MySQL*** server like this:
```
shell62; cd mysql_installation_directory
shell62; bin/safe_mysqld 38;
```

   If you have problems starting the server, see 4.15.2 Problèmes avec le serveur ***MySQL***.

3. Use `mysqladmin` to verify that the server is running. The following commands provide a simple test to check that the server is up and responding to connections:
```
shell62; BINDIR/mysqladmin version
shell62; BINDIR/mysqladmin variables
```

   The output from `mysqladmin version` varies slightly depending on your platform and version of ***MySQL***, but should be similar to that shown below:

```
shell62; BINDIR/mysqladmin version
mysqladmin  Ver 6.3 Distrib 3.22.9−beta, for pc−linux−gnu on i686
TCX Datakonsult AB, by Monty

Server version          3.22.9−beta
Protocol version        10
Connection              Localhost via UNIX socket
TCP port                3306
UNIX socket             /tmp/mysql.sock
Uptime:                 16 sec

Running threads: 1  Questions: 20  Reloads: 2  Open tables: 3
```

   To get a feeling for what else you can do with `BINDIR/mysqladmin`, invoke it with the `−−help` option.

4. Verify that you can shut down the server:
```
shell62; BINDIR/mysqladmin −u root shutdown
```

5. Verify that you can restart the server. Do this using `safe_mysqld` or by invoking `mysqld` directly. For example:
```
shell62; BINDIR/safe_mysqld −−log 38;
```

If `safe_mysqld` fails, try running it from the ***MySQL*** installation directory (if you are not already there). If that doesn't work, see 4.15.2 Problèmes avec le serveur ***MySQL***.

6. Run some simple tests to verify that the server is working. The output should be similar to what is shown below:

```
shell62; BINDIR/mysqlshow
+−−−−−−−−−−−+
| Databases |
+−−−−−−−−−−−+
| mysql     |
+−−−−−−−−−−−+

shell62; BINDIR/mysqlshow mysql
Database: mysql
+−−−−−−−−−−−−−−+
|    Tables    |
+−−−−−−−−−−−−−−+
| columns_priv |
| db           |
| func         |
| host         |
| tables_priv  |
| user         |
+−−−−−−−−−−−−−−+

shell62; BINDIR/mysql −e "select host,db,user from db" mysql
+−−−−−−+−−−−−−−−+−−−−−−+
| host | db     | user |
+−−−−−−+−−−−−−−−+−−−−−−+
| %    | test   |      |
| %    | test_% |      |
+−−−−−−+−−−−−−−−+−−−−−−+
```

There is also a benchmark suite in the `sql−bench'` directory (under the ***MySQL*** installation directory) that you can use to compare how ***MySQL*** performs on different platforms. The `sql−bench/Results'` directory contains the results from many runs against different databases and platforms. To run all tests, execute these commands:

```
shell62; cd sql−bench
shell62; run−all−tests
```

If you don't have the `sql−bench'` directory, you are probably using an RPM for a binary distribution. (Source distribution RPMs include the benchmark directory.) In this case, you must first install the benchmark suite before you can use it. Beginning with ***MySQL*** 3.22, there are benchmark RPM files named `mysql−bench−VERSION−i386.rpm'` that contain benchmark code and data. If you have a source distribution, you can also run the tests in the `tests'` subdirectory. For example, to run `auto_increment.tst'`, do this:

```
shell62; BINDIR/mysql −vvf test 60; ./tests/auto_increment.tst
```

The expected results are shown in the `./tests/auto_increment.res'` file.

4.15.1 Problèmes avec `mysql_install_db`

This section lists problems you might encounter when you run `mysql_install_db`:

`mysql_install_db` doesn't install the grant tables

You may find that `mysql_install_db` fails to install the grant tables and terminates after displaying the following messages:

```
starting mysqld daemon with databases from XXXXXX
mysql daemon ended
```

In this case, you should examine the log file very carefully! The log should be located in the directory `XXXXXX'` named by the error message, and should indicate why `mysqld` didn't start. If you don't understand what happened, include the log when you post a bug report using `mysqlbug`! 2.3 Comment rapporter des bugs et des problèmes.

There is already a `mysqld` daemon running

In this case, you have probably don't have to run `mysql_install_db` at all. You have to run `mysql_install_db` only once, when you install ***MySQL*** the first time.

Installing a second `mysqld` daemon doesn't work when one daemon is running

This can happen when you already have an existing ***MySQL*** installation, but want to put a new installation in a different place (e.g., for testing, or perhaps you simply want to run two installations at the same time). Generally the problem that occurs when you try to run the second server is that it tries to use the same socket and port as the old one. In this case you will get the error message: `Can't start server: Bind on TCP/IP port: Address already in use` or `Can't start server : Bind on unix socket...` You can start the new server with a different socket and port as follows:

```
shell62; MYSQL_UNIX_PORT=/tmp/mysqld−new.sock
shell62; MYSQL_TCP_PORT=3307
shell62; export MYSQL_UNIX_PORT MYSQL_TCP_PORT
shell62; scripts/mysql_install_db
shell62; bin/safe_mysqld 38;
```

After this, you should edit your server boot script to start both daemons with different sockets and ports. For example, it could invoke `safe_mysqld` twice, but with different `−−socket`, `−−port` and `−−basedir` options for each invocation.

You don't have write access to `/tmp'

If you don't have write access to create a socket file at the default place (in `/tmp'`) or permission to create temporary files in `/tmp,'` you will get an error when running `mysql_install_db` or when starting or using `mysqld`. You can specify a different socket and temporary directory as follows:

```
shell62; TMPDIR=/some_tmp_dir/
shell62; MYSQL_UNIX_PORT=/some_tmp_dir/mysqld.sock
shell62; export TMPDIR MYSQL_UNIX_PORT
```

`some_tmp_dir'` should be the path to some directory for which you have write permission. After this you should be able to run `mysql_install_db` and start the server with these commands:

```
shell62; scripts/mysql_install_db
shell62; BINDIR/safe_mysqld 38;
```

`mysqld` crashes immediately

If you are running RedHat 5.0 with a version of `glibc` older than 2.0.7−5, you should make sure you have installed all `glibc` patches! There is a lot of information about this in the ***MySQL*** mail archives. Links to the mail archives are available at the online ***MySQL*** documentation page. Also, see 4.11.5 Linux (Toutes versions de Linux). You can also start `mysqld` manually using the `−−skip−grant` option and add the privilege information yourself using `mysql`:

```
shell62; BINDIR/safe_mysqld −−skip−grant 38;
shell62; BINDIR/mysql −u root mysql
```

From `mysql`, manually execute the SQL commands in `mysql_install_db`. Make sure you run `mysqladmin reload` afterward to tell the server to reload the grant tables.

4.15.2 Problèmes avec le serveur *MySQL*

Generally, you start the `mysqld` server in one of three ways:

- By invoking `mysql.server`. This script is used primarily at system startup and shutdown, and is described more fully in 4.15.3 Démarrer et arrête ***MySQL*** automatiquement.
- By invoking `safe_mysqld`, which tries to determine the proper options for `mysqld` and then runs it with those options.
- By invoking `mysqld` directly.

Whichever méthode you use to start the server, if it fails to start up correctly, check the log file to see if you can find out why. Log files are located in the data directory (typically `/usr/local/mysql/data'` for a binary distribution, `/usr/local/var'` for a source distribution). Look in the data directory for files with names of the form `host_name.err'` and `host_name.log'` where `host_name` is the name of your server host. Then check the last few lines of these files:

```
shell62; tail host_name.err
shell62; tail host_name.log
```

When the `mysqld` daemon starts up, it changes directory to the data directory. This is where it expects to write log files and the pid (process ID) file, and where it expects to find databases.

The data directory location is hardwired in when the distribution is compiled. However, if `mysqld` expects to find the data directory somewhere other than where it really is on your system, it will not work properly. If you have problems with incorrect paths, you can find out what options `mysqld` allows and what the default path settings are by invoking `mysqld` with the `−−help` option. You can override the defaults by specifying the correct pathnames as command−line arguments to `mysqld`. (These options can be used with `safe_mysqld` as well.)

Normally you should need to tell `mysqld` only the base directory under which ***MySQL*** is installed. You can do this with the `−−basedir` option. You can also use `−−help` to check the effect of changing path options (note that `−−help` *must* be the final option of the `mysqld` command). For example:

```
shell62; EXECDIR/mysqld −−basedir=/usr/local −−help
```

Once you determine the path settings you want, start the server without the `−−help` option.

If you get the following error, it means that some other program (or another `mysqld` server) is already using the TCP/IP port or socket `mysqld` is trying to use:

```
Can't start server: Bind on TCP/IP port: Address already in use
  or
Can't start server : Bind on unix socket...
```

Use `ps` to make sure that you don't have another `mysqld` server running. If you can't find another server running, you can try to execute the command `telnet your−host−name tcp−ip−port−number` and press `RETURN` a couple of times. If you don't get a error message like `telnet: Unable to connect to remote host: Connection refused`, something is using the TCP/IP port `mysqld` is trying to use. `mysql_install_db`, and 19.3 Faire tourner plusieurs serveurs MySQL sur la même machine.

The `safe_mysqld` script is written so that it normally is able to start a server that was installed from either a source or a binary version of ***MySQL***, even if these install the server in slightly different locations. `safe_mysqld` expects one of these conditions to be true:

- The server and databases can be found relative to the directory from which `safe_mysqld` is invoked. `safe_mysqld` looks under its working directory for `` `bin' `` and `` `data' `` directories (for binary distributions) or for `` `libexec' `` and `` `var' `` directories (for source distributions). This condition should be met if you execute `safe_mysqld` from your ***MySQL*** installation directory (for example, `` `/usr/local/mysql' `` for a binary distribution).
- If the server and databases cannot be found relative to its working directory, `safe_mysqld` attempts to locate them by absolute pathnames. Typical locations are `` `/usr/local/libexec' `` and `` `/usr/local/var' ``. The actual locations are determined when the distribution was built from which `safe_mysqld` comes. They should be correct if ***MySQL*** was installed in a standard location.

Since `safe_mysqld` will try to find the server and databases relative to its own working directory, you can install a binary distribution of ***MySQL*** anywhere, as long as you start `safe_mysqld` from the ***MySQL*** installation directory:

```
shell62; cd mysql_installation_directory
shell62; bin/safe_mysqld 38;
```

If `safe_mysqld` fails, even when invoked from the ***MySQL*** installation directory, you can modify it to use the path to `mysqld` and the pathname options that are correct for your system. Note that if you upgrade ***MySQL*** in the future, your modified version of `safe_mysqld` will be overwritten, so you should make a

copy of your edited version that you can reinstall.

If `mysqld` is currently running, you can find out what path settings it is using by executing this command:

```
shell62; mysqladmin variables
```

or

```
shell62; mysqladmin −h 'your−host−name' variables
```

If `safe_mysqld` starts the server but you can't connect to it, you should make sure you have an entry in `` `/etc/hosts' `` that looks like this:

```
127.0.0.1       localhost
```

This problem occurs only on systems that don't have a working thread library and for which ***MySQL*** must be configured to use MIT−pthreads.

4.15.3 Démarrer et arrête *MySQL* automatiquement

The `mysql.server` script can be used to start or stop the server, by invoking it with `start` or `stop` arguments:

```
shell62; mysql.server start
shell62; mysql.server stop
```

`mysql.server` can be found in the `` `share/mysql' `` directory under the ***MySQL*** installation directory, or in the `` `support−files' `` directory of the ***MySQL*** source tree.

Before `mysql.server` starts the server, it changes directory to the ***MySQL*** installation directory, then invokes `safe_mysqld`. You might need to edit `mysql.server` if you have a binary distribution that you've installed in a non−standard location. Modify it to `cd` into the proper directory before it runs `safe_mysqld`. If you want the server to run as some specific user, you can change the `mysql_daemon_user=root` line to use another user. You can also modify `mysql.server` to pass other options to `safe_mysqld`.

`mysql.server stop` brings down server by sending a signal to it. You can take down the server manually by executing `mysqladmin shutdown`.

You might want to add these start and stop commands to the appropriate places in your `` `/etc/rc*' `` files when you start using ***MySQL*** for production applications. Note that if you modify `mysql.server`, then if you upgrade ***MySQL*** sometime, your modified version will be overwritten, so you should make a copy of your edited version that you can reinstall.

If your system uses `` `/etc/rc.local' `` to start external scripts, you should append the following to it:

```
/bin/sh −c 'cd /usr/local/mysql ; ./bin/safe_mysqld 38;'
```

You can also add options or `mysql.server` in a global `` `/etc/my.cnf' `` file. A typical `` `/etc/my.cnf' `` file might look like this:

```
[mysqld]
```

```
datadir=/usr/local/mysql/var
socket=/tmp/mysqld.sock
port=3306

[mysql.server]
user=mysql
basedir=/usr/local/mysql
```

The `mysql.server` script uses the following variables: `user`, `datadir`, `basedir`, `bindir` and `pid−file`.

4.15.4 Fichier d'options.

4.15.4 Fichier d'options

MySQL 3.22 can read default startup options for the server and for clients from option files.

MySQL reads default options from the following files on Unix:

Filename	*Purpose*
`/etc/my.cnf`	Global options
`DATADIR/my.cnf`	Server−specific options
`~/.my.cnf`	User−specific options

`DATADIR` is the ***MySQL*** data directory (typically `` `/usr/local/mysql/data' `` for a binary installation, or `` `/usr/local/var' `` for a source installation). Note that this is the directory that was specified at configuration time, not the one specified with `−−datadir` when `mysqld` starts up! (`−−datadir` has no effect on where the server looks for option files, because it looks for them before it processes any command−line arguments.)

MySQL reads default options from the following files on Win32:

Filename	*Purpose*
`windows−system−directory\my.ini`	
`C:\my.cnf`	Global options
`C:\mysql\data\my.cnf`	Server−specific options

Note that you on Win32 should specify all paths with `/` instead of `\`. If you use `\`, you need to specify this twice, as `\` is the escape character in ***MySQL***.

MySQL tries to read option files in the order listed above. If multiple option files exist, an option specified in a file read later takes precedence over the same option specified in a file read earlier. Options specified on the command line take precedence over options specified in any option file. Some options can be specified using environment variables. Options specified on the command line or in option files take precedence over environment variable values.

The following programs support option files: `mysql`, `mysqladmin`, `mysqld`, `mysqldump`, `mysqlimport`, `mysql.server`, `isamchk` and `pack_isam`.

You can use option files to specify any long option that a program supports! Run the program with `--help` to get a list of available options.

An option file can contain lines of the following forms:

`#comment`
: Comment lines starts with `` `#' `` or `` `;' ``. Empty lines are ignored.

`[group]`
: `group` is the name of the program or group for which you want to set options. After a group line, any `option` or `set-variable` lines apply to the named group until the end of the option file or another group line is given.

`option`
: This is equivalent to `--option` on the command line.

`option=value`
: This is equivalent to `--option=value` on the command line.

`set-variable = variable=value`
: This is equivalent to `--set-variable variable=value` on the command line. This syntax must be used to set a `mysqld` variable.

The `client` group allows you to specify options that apply to all ***MySQL*** clients (not `mysqld`). This is the perfect group to use to specify the password you use to connect to the server. (But make sure the option file is readable and writable only to yourself.)

Note that for options and values, all leading and trailing blanks are automatically deleted. You may use the escape sequences `` `\b' ``, `` `\t' ``, `` `\n' ``, `` `\r' ``, `` `\\' `` and `` `\s' `` in your value string (`` `\s' `` == blank).

Here is a typical global option file:

```
[client]
port=3306
socket=/tmp/mysql.sock

[mysqld]
port=3306
socket=/tmp/mysql.sock
set-variable = key_buffer=16M
set-variable = max_allowed_packet=1M

[mysqldump]
quick
```

Here is typical user option file:

```
[client]
# The following password will be sent to all standard MySQL clients
password=my_password

[mysql]
no-auto-rehash
```

If you have a source distribution, you will find a sample configuration file named `` `my-example.cnf' `` in the `` `support-files' `` directory. If you have a binary distribution, look in the `` `DIR/share/mysql' `` directory, where `DIR` is the pathname to the ***MySQL*** installation directory (typically `` `/usr/local/mysql' ``). You can copy `` `my-example.cnf' `` to your home directory (rename the copy to `` `.my.cnf' ``) to experiment with.

To tell a ***MySQL*** program not to read any option files, specify `--no-defaults` as the first option on the command line. This ***MUST*** be the first option or it will have no effect! If you want to check which options are used, you can give the option `--print-defaults` as the first option.

If you want to force the use of a specific config file, you can use the option `−−defaults−file=full−path−to−default−file`. If you do this, only the specified file will be read.

Note for developers: Option file handling is implemented simply by processing all matching options (i.e., options in the appropriate group) before any command line arguments. This works nicely for programs that use the last instance of an option that is specified multiple times. If you have an old program that handles multiply−specified options this way but doesn't read option files, you need add only two lines to give it that capability. Check the source code of any of the standard ***MySQL*** clients to see how to do this.

4.16 Y a t il des manipulations particulières lors de la mise à jour?

You can always move the ***MySQL*** form and data files between different versions on the same architecture as long as you have the same base version of ***MySQL***. The current base version is 3. If you change the character set by recompiling ***MySQL*** (which may also change the sort order), you must run `isamchk −r −q` on all tables. Otherwise your indexes may not be ordered correctly.

If you are paranoid and/or afraid of new versions, you can always rename your old `mysqld` to something like `mysqld−`'old−version−number'. If your new `mysqld` then does something unexpected, you can simply shut it down and restart with your old `mysqld`!

When you do an upgrade you should also backup your old databases, of course. Sometimes it's good to be a little paranoid!

After an upgrade, if you experience problems with recompiled client programs, like `Commands out of sync` or unexpected core dumps, you probably have used an old header or library file when compiling your programs. In this case you should check the date for your `` `mysql.h' `` file and `` `libmysqlclient.a' `` library to verify that they are from the new ***MySQL*** distribution. If not, please recompile your programs!

If you get some problems that the new `mysqld` server doesn't want to start or that you can't connect without a password, check that you don't have some old `` `my.cnf' `` file from your old installation! You can check this with: `program−name −−print−defaults`. If this outputs anything other than the program name, you have a active `my.cnf` file that will may affect things!

It is a good idea to rebuild and reinstall the `Msql−Mysql−modules` distribution whenever you install a new release of ***MySQL***, particularly if you notice symptoms such as all your `DBI` scripts dumping core after you upgrade ***MySQL***.

4.16.1 Mise à jour de a 3.22 vers 3.23

MySQL 3.23 supports tables of the new `MyISAM` type and the old `NISAM` type. You don't have to convert your old tables to use these with 3.23. By default, all new tables will be created with type `MyISAM` (unless you start `mysqld` with the `−−default−table−type=isam` option. You can change an `ISAM` table to a `MyISAM` table with `ALTER TABLE` or the Perl script `mysql_convert_table_format`.

3.22 and 3.21 clients will work without any problems with a 3.23 server.

The following lists what you have to watch out for when upgrading to 3.23:

- `INNER` and `DELAYED` are now reserved words.
- `FLOAT(4)` and `FLOAT(8)` are now true floating point types.
- When declaring `DECIMAL(length,dec)` the length argument no longer includes a place for the sign or the decimal point.
- A `TIME` string must now be of one of the following formats: `[[[DAYS] [H]H:]MM:]SS[.fraction]` or `[[[[[H]H]H]H]MM]SS[.fraction]`
- `LIKE` now compares strings using the same character comparison rules as `'='`. If you require the old behavior, you can compile ***MySQL*** with the `CXXFLAGS=−DLIKE_CMP_TOUPPER` flag.
- When you check/repair tables you should use `myisamchk` for `MyISAM` tables (`.MYI`) and `isamchk` for ISAM (`.ISM`) tables.
- If you want your `mysqldump`s to be compatible between ***MySQL*** 3.22 and 3.23, you should not use the `−−opt` or `−−full` option to `mysqldump`.
- Check all your calls to `DATE_FORMAT()` to make sure there is a `` `%' `` before each format character.
- `mysql_fetch_fields_direct` is now a function (it was a macro) and it returns a pointer to a `MYSQL_FIELD` instead of a `MYSQL_FIELD`.
- `mysql_num_fields()` can no longer be used on a `MYSQL*` object (it's now a function that takes `MYSQL_RES*` as an argument. You should now use `mysql_field_count()` instead.
- In `MySQL` 3.22, the output of `SELECT DISTINCT ...` was almost always sorted. In 3.23, you must use `GROUP BY` or `ORDER BY` to obtain sorted output.
- `SUM()` now returns `NULL`, instead of 0, if there is no matching rows. This is according to ANSI SQL.
- New restricted words: `CASE, THEN, WHEN, ELSE and END`

4.16.2 Mise à jour de a 3.21 vers 3.22

Nothing that affects compatibility has changed between 3.21 and 3.22. The only pitfall is that new tables that are created with `DATE` type columns will use the new way to store the date. You can't access these new fields from an old version of `mysqld`.

After installing ***MySQL*** 3.22, you should start the new server and then run the `mysql_fix_privilege_tables` script. This will add the new privileges that you need to use the `GRANT` command. If you forget this, you will get `Access denied` when you try to use `ALTER TABLE`, `CREATE INDEX` or `DROP INDEX`. If your ***MySQL*** root user requires a password, you should give this as an argument to `mysql_fix_privilege_tables`.

The C API interface to `mysql_real_connect()` has changed. If you have an old client program that calls this function, you must place a `0` for the new `db` argument (or recode the client to send the `db` element for faster connections). You must also call `mysql_init()` before calling `mysql_real_connect()`! This change was done to allow the new `mysql_options()` function to save options in the `MYSQL` handler structure.

4.16.3 Mise à jour de a 3.20 vers 3.21

If you are running a version older than 3.20.28 and want to switch to 3.21.x, you need to do the following:

You can start the `mysqld` 3.21 server with `safe_mysqld −−old−protocol` to use it with clients from the 3.20 distribution. In this case, the new client function `mysql_errno()` will not return any server error, only `CR_UNKNOWN_ERROR`, (but it works for client errors) and the server uses the old password() checking rather than the new one.

If you are ***NOT*** using the `−−old−protocol` option to `mysqld`, you will need to make the following changes:

- All client code must be recompiled. If you are using ODBC, you must get the new ***MyODBC*** 2.x driver.
- The script `scripts/add_long_password` must be run to convert the `Password` field in the `mysql.user` table to `CHAR(16)`.
- All passwords must be reassigned in the `mysql.user` table (to get 62−bit rather than 31−bit passwords).

- The table format hasn't changed, so you don't have to convert any tables.

MySQL 3.20.28 and above can handle the new `user` table format without affecting clients. If you have a ***MySQL*** version earlier than 3.20.28, passwords will no longer work with it if you convert the `user` table. So to be safe, you should first upgrade to at least 3.20.28 and then upgrade to 3.21.x.

The new client code works with a 3.20.x `mysqld` server, so if you experience problems with 3.21.x, you can use the old 3.20.x server without having to recompile the clients again.

If you are not using the `--old-protocol` option to `mysqld`, old clients will issue the error message:

```
ERROR: Protocol mismatch. Server Version = 10 Client Version = 9
```

The new Perl `DBI`/`DBD` interface also supports the old `mysqlperl` interface. The only change you have to make if you use `mysqlperl` is to change the arguments to the `connect()` function. The new arguments are: `host`, `database`, `user`, `password` (the `user` and `password` arguments have changed places). Perl `DBI` Class.

The following changes may affect requêtesin old applications:

- `HAVING` must now be specified before any `ORDER BY` clause.
- The parameters to `LOCATE()` have been swapped.
- There are some new reserved words. The most notable are `DATE`, `TIME` and `TIMESTAMP`.

4.16.4 Mise à jour vers une autre architecture

If you are using ***MySQL*** 3.23, you can copy the `.frm`, the `.MYI` and the `.MYD` files between different architectures that support the same floating point format. (***MySQL*** takes care of any byte swapping issues).

The ***MySQL*** data `` `*.ISD' `` and the index files `` `*.ISM' `` files) are architecture−dependent and in some case OS−dependent. If you want to move your applications to another machine that has a different architecture or OS than your current machine, you should not try to move a database by simply copying the files to the other machine. Use `mysqldump` instead.

By default, `mysqldump` will create a file full of SQL statements. You can then transfer the file to the other machine and feed it as input to the `mysql` client.

Try `mysqldump --help` to see what options are available. If you are moving the data to a newer version of ***MySQL***, you should use `mysqldump --opt` with the newer version to get a fast, compact dump.

The easiest (although not the fastest) way to move a database between two machines is to run the following commands on the machine on which the database is located:

```
shell62; mysqladmin −h 'other hostname' create nom_base_de_donnees
shell62; mysqldump −−opt nom_base_de_donnees \
        | mysql −h 'other hostname' nom_base_de_donnees
```

If you want to copy a database from a remote machine over a slow network, you can use:

```
shell62; mysqladmin create nom_base_de_donnees
shell62; mysqldump −h 'other hostname' −−opt −−compress nom_base_de_donnees \
        | mysql nom_base_de_donnees
```

You can also store the result in a file, then transfer the file to the target machine and load the file into the database there. For example, you can dump a database to a file on the source machine like this:

```
shell62; mysqldump −−quick nom_base_de_donnees | gzip 62; nom_base_de_donnees.contents.gz
```

(The file created in this example is compressed.) Transfer the file containing the database contents to the target machine and run these commands there:

```
shell62; mysqladmin create nom_base_de_donnees
shell62; gunzip 60; nom_base_de_donnees.contents.gz | mysql nom_base_de_donnees
```

You can also use `mysqldump` and `mysqlimport` to accomplish the database transfer. For big tables, this is much faster than simply using `mysqldump`. In the commands shown below, `DUMPDIR` represents the full pathname of the directory you use to store the output from `mysqldump`.

First, create the directory for the output files and dump the database:

```
shell62; mkdir DUMPDIR
shell62; mysqldump −−tab=DUMPDIR nom_base_de_donnees
```

Then transfer the files in the `DUMPDIR` directory to some corresponding directory on the target machine and load the files into ***MySQL*** there:

```
shell62; mysqladmin create nom_base_de_donnees           # create database
shell62; cat DUMPDIR/*.sql | mysql nom_base_de_donnees   # create tables in database
shell62; mysqlimport nom_base_de_donnees DUMPDIR/*.txt   # load data into tables
```

Also, don't forget to copy the `mysql` database, since that's where the grant tables (`user`, `db`, `host`) are stored. You may have to run commands as the ***MySQL*** `root` user on the new machine until you have the `mysql` database in place.

After you import the `mysql` database on the new machine, execute `mysqladmin flush−privileges` so that the server reloads the grant table information.

5 Compatibilité avec les standards

5.1 Extensions au langage ANSI SQL92

MySQL apporte quelques extensions à la norme ANSI SQL 92 que vous ne trouverez probablement dans aucune autre base de données. Soyez conscient que si vous utilisez ces extensions, votre code ne sera pas portable sur une autre base SQL. Dans certains cas, vous pouvez écrire du code qui utilise ces extensions ***MySQL***, mais qui est portable, en utilisant les commentaires de la forme `/*! ... */`. Dans ce cas, ***MySQL*** va analyser puis exécuter le code de ce commentaire comme n'importe quelle commande ***MySQL***, mais les autres serveur SQL les ignoreront. Par exemple :

```
SELECT /*! STRAIGHT_JOIN */ nom_colonne FROM table1,table2 WHERE ...
```

Voici la liste des extensions ***MySQL***:

- Les types de colonnes `MEDIUMINT`, `SET`, `ENUM` et tous les types `BLOB` et `TEXT`.
- Les attributs de champs `AUTO_INCREMENT`, `BINARY`, `UNSIGNED` et `ZEROFILL`.
- Toutes les comparaisons de chaînes sont insensibles à la casse par défaut, et le classement des chaînes est basé sur la table de caractère courante (par défaut, ISO−8859−1 Latin1). Si vous souhaitez que ces opérations soient sensible à la casse, ajoutez l'attribut `BINARY` aux colonnes concernés, ou bien utiliser l'opérateur de transptypage `BINARY` lors des comparaisons pour qu'elles prennent en compte l'ordre ASCII en cours sur l'hôte ***MySQL***.
- ***MySQL*** fait correspondre à chaque base de données un dossier dans le dossier de données ***MySQL***, et les tables de cette base seront représentées par des fichiers. Cela a deux conséquences :
  - Les noms de bases de donnés et de tables sont dépendants du système d'exploitation sur lequel ***MySQL*** tourne, notamment concertant la casse des noms. Si vous avez des difficultés à vous souvenir des noms de tables, utilisez une convention cohérente, comme par exemple celle de mettre tous les noms en minuscules.
  - Les noms de bases, de table, d'index, de colonne ou d' alias peuvent commencer par un chiffre (mais de doivent pas être consistitués seulement de chiffres).
  - Vous pouvez utiliser les commandes du système pour sauvegarder, renommer, déplacer, effacer et copier des tables. Par exemple, vous pouvez copier le ``.ISD'', ``.ISM'' et ``.frm'' pour créer une nouvelle table.
- En SQL pur, vous pouvez accéder à une table d'une autre base de données avec la syntaxe : `nom_base_de_donnees.nom_table`. Certains serveurs SQL propose la même fonctionnalité, mais l'appelle `User space`. ***MySQL*** n'accepte pas les espaces de tables comme dans : `create table ralph.my_table...IN my_tablespace`.
- `LIKE` est utilisable avec les colonnes de type numérique
- Possibilité d'utilisation de `INTO OUTFILE` et `STRAIGHT_JOIN` dans les commandes `SELECT`. SELECT.
- L'option `SQL_SMALL_RESULT` dans une commande `SELECT`.
- `EXPLAIN SELECT` permet de voir de quelle manière les tables ont été regroupées.
- L'utilisation des index de noms, ou des index sur le préfixe d'un champs et l'utilisation de `INDEX` ou `KEY` dans une commande `CREATE TABLE`. CREATE TABLE
- Attribut `TEMPORARY` et `IF NOT EXISTS` dans la commande `CREATE TABLE`.
- Utilisation de `COUNT(DISTINCT liste)` lorsque 'liste' contient plus d'un élément.
- Utilisation de `CHANGE nom_colonne`, `DROP nom_colonne` ou `DROP INDEX` dans les commandes `ALTER TABLE`. ALTER TABLE.
- Utilisation de `IGNORE` dans les commandes `ALTER TABLE`.
- Utilisation de multiples `ADD`, `ALTER`, `DROP` ou `CHANGE` dans les commandes `ALTER TABLE`.
- Utilisation de `DROP TABLE` avec les mots clés `IF EXISTS`.
- Vous pouvez supprimer plusieurs tables avec une seule commande `DROP TABLE`.
- Clause `LIMIT` dans les commandes `DELETE`.
- L'option `DELAYED` dans les commandes `INSERT` et `REPLACE`.
- L'option `LOW_PRIORITY` dans les commandes `INSERT`, `REPLACE`, `DELETE` et `UPDATE`.
- Utilisation de `LOAD DATA INFILE`. Dans certains cas, cette syntaxe est compatibles avec la fonction `LOAD DATA INFILE` d'Oracle. LOAD DATA.
- La commande `OPTIMIZE TABLE`. OPTIMIZE TABLE.
- La commande `SHOW`. SHOW.
- Les chaînes de caractères peuvent être définies avec ``"'' ou ``'', et pas seulement avec ``''.
- Utilisation du caractère d'échappement ``\''.
- La commande `SET OPTION`. SET OPTION.
- Vous n'avez pas à nommer expressément toutes les colonnes dans une clause `GROUP BY`. Mais cela peut permettre d'accélérer quelques requêtes très spécifiques. 7.3.13 Fonctions à utiliser dans les clauses GROUP BY.
- Pour aider les utilisateurs d'autres environnements SQL, ***MySQL*** dispose d'aliases pour de nombreuses fonctions. Par exemple, toutes les fonctions sur les chaînes de caractère supporte la syntaxe ANSI SQL et ODBC.
- ***MySQL*** accepte les opérateurs logiques `||` et (OR et AND, comme en langage C). Sous ***MySQL***, `||` et `OR` sont synonymes, tout comme le sont `&et AND. A cause de ce double emploi, MySQL n'accepte pas les op&eacute;rateurs ANSI SQL`

`||` comme additionneur de chaînes; (utilisez à la place `CONCAT()`).. Etant donné que `CONCAT()` prend un nombre arbitraire d'arguments, il est facile de remplacer `||`.

- `CREATE DATABASE` ou `DROP DATABASE`. `CREATE DATABASE`
- L'opérateur `%` est synonyme de `MOD()`. C'est à dire que, `N % M` est équivalent à `MOD(N,M)`. `%` est hérité du langage C, et permet la compatibilité avec PostgreSQL.
- Les opérateurs =, <>, <= ,<, >=,>, <<, >>, <=>, `AND`, `OR` ou `LIKE` peuvent être utilisés dans les comparaisons de colonnes avec la clause `FROM` dans les commandes `SELECT`. Par exemple :

```
mysql62; SELECT col1=1 AND col2=2 FROM nom_table;
```

- La fonction `LAST_INSERT_ID()`. `mysql_insert_id`.
- Les opérateurs d'expressions régulières `REGEXP` et `NOT REGEXP`.
- `CONCAT()` ou `CHAR()` avec plus d'un one argument. (Avec ***MySQL***, ces fonctions peuvent prendre un nombre arbitraire d'arguments).
- Les fonctions supplémentaires `BIT_COUNT()`, `ELT()`, `FROM_DAYS()`, `FORMAT()`, `IF()`, `PASSWORD()`, `ENCRYPT()`, `md5()`, `ENCODE()`, `DECODE()`, `PERIOD_ADD()`, `PERIOD_DIFF()`, `TO_DAYS()`, ou `WEEKDAY()`.
- Utilisation de `TRIM()` pour supprimer les espaces de début et fin de chaîne. ANSI SQL ne supporte que la suppression de caractères uniques.
- Les fonctions de `GROUP BY` : `STD()`, `BIT_OR()` et `BIT_AND()`.
- Utilisation de `REPLACE` à la place de `DELETE` + `INSERT`. `REPLACE`.
- La commande `FLUSH flush_option`.

5.2 Différences entre MySQL et ANSI SQL92

Nous t chons de mettre ***MySQL*** aux normes ANSI SQL et ODBC SQL, mais dans certains cas, ***MySQL*** gère les choses de manière différente.

- `−−` n'est pas un commentaire. 5.3.7 `` `−−' `` comme début de commentaire.
- Pour les colonnes de type `VARCHAR`, les espaces de fin de chaînes sont supprimés à l'enregistrement. `Bugs`.
- Dans certains cas, les colonnes de type `CHAR` sont spontanément changées en colonne de type `VARCHAR`. 7.6.1 Modifications automatiques de type de colonne.
- Les droits d'une table ne sont pas supprimés lorsque vous supprimez une table. Vous devez explicitement appeler la commande `REVOKE` pour supprimer les droits de la table. `GRANT`.

5.3 Functionalités manquantes

Les fonctionnalités suivantes manquent dans la version courante de ***MySQL***. La liste des fonctionnalités classées par ordre de priorité est disponible en ligne à : F Liste de voeux pour les versions futures de MySQL (la TODO).

5.3.1 Sub−select

La requête suivante ne fonctionne par encore sous ***MySQL***:

```
SELECT * FROM table1 WHERE id IN (SELECT id FROM table2);
SELECT * FROM table1 WHERE id NOT IN (SELECT id FROM table2);
```

Cependant, il est souvent possible de se passer d'une sous sélection :

```
SELECT table1.* FROM table1,table2 WHERE table1.id=table2.id;
SELECT table1.* FROM table1 LEFT JOIN table2 ON table1.id=table2.id where table2.id IS NULL
```

Pour les sous requêtes compliquées, vous pouvez toujours créer une table temporaire, et y appliquer votre requête.

MySQL ne supporte que `INSERT ... SELECT ...` et `REPLACE ... SELECT ...` Les sous sélections indépendantes ne seront disponibles qu'à partir de la version 3.24.0. Actuellement, vous pouvez toujours utiliser la fonction `IN()` dans d'autres contextes.

5.3.2 SELECT INTO TABLE

MySQL ne supporte pas encore cette extension du SQL Orable.: `SELECT ... INTO TABLE ...`. A la place, ***MySQL*** supporte la syntaxe de la norme ANSI SQL `INSERT INTO ... SELECT ...`, ce qui est pratiquement la même chose.

Alternativement, vous pouvez utiliser `SELECT INTO OUTFILE...` ou `CREATE TABLE ... SELECT` pour résoudre votre problème.

5.3.3 Transactions

Les transactions ne sont pas encore supportées par ***MySQL***. Le serveur va bientôt accepter des opérations atomiques, ce qui permettra des transactions, mais sans le rollback. Avec les opérations atomiques, vous pourrez exécuter des groupes de commandes `INSERT`/`SELECT`/ avec n'importe quelle commande, et être sur qu'il n'y aura aucune interférence avec un autre thread. Dans ce contexte, vous n'aurez alors pas besoin de rollback. Actuellement, vous pouvez empêcher les autres threads d'interférer en utilisant les fonctions `LOCK TABLES` et `UNLOCK TABLES`. `LOCK TABLES`.

5.3.4 Procédures stockées et triggers

Une fonctions enregistrée est un ensemble de commandes SQL qui peut être compilé et enregistré sur le serveur. Une fois fait, les clients peuvent se référer à cette fonction pour exécuter l'ensemble des commandes. Cela accélère le traitement des requêtes, car elles n'ont pas a être analysées, et moins d'information circule entre le client et le serveur. Il est aussi possible d'élever le niveau de conception, en b tissant des bibliothèques.

Un trigger est une fonction enregistrée qui est invoquées à chaque fois qu'un événement particulier survient. Par exemple, vous pourriez installer une fonction qui sera lancée à chaque fois qu'un enregistrement sera effacé dans une table de transaction, pour effacer automatiquement les informations correspondantes dans les tables de clients.

Lors de modifications ultérieures, MySQL sera capable de gérer les fonctions enregistrées, mais pas les triggers. En général, les triggers ralentissent le serveur, même pour des requêtes pour lesquelles ils ne sont pas appelés.

Pour savoir quand ***MySQL*** disposera des procédures enregistrées, sinon, F Liste de voeux pour les versions futures de MySQL (la TODO).

5.3.5 Clés étrangères

Remarque : les clés externes en SQL ne sont pas utilisées pour effectuer des regroupements de table, mais pour assurer l'intégrité référentielle. Si vous voulez lire des informations depuis plusieurs tables avec une commande `SELECT`, c'est un regroupement !

```
SELECT * from table1,table2 where table1.id = table2.id;
```

`JOIN`. 8.3.5 Utiliser des clés étrangères

La syntaxe `FOREIGN KEY` de ***MySQL*** n'existe que pour la compatibilité avec les commandes `CREATE TABLE` des autres bases SQL : elle n'a aucun effet. La syntaxe `FOREIGN KEY` sans la partie `ON DELETE`

`...` n'est utilisée que pour des raisons de documentation. Certaines applications ODBC l'utilise pour générer des clauses `WHERE` automatiques, mais il est généralement simple à remplacer. `FOREIGN KEY` est parfois utilisé comme contrainte, mais ce genre de vérification n'est pas nécessaire si les lignes ont été insérées dans l'ordre. ***MySQL*** ne supporte que ces clauses, car certaines applications en ont besoin (qu'elle fonctionne ou pas).

Avec ***MySQL***, vous pouvez contourner le problème sans la clause `ON DELETE ...` en ajoutant une commande `DELETE` adéquate lors de l'effacement d'un enregsitrement d'une table qui a une clé externe. En pratique, c'est aussi rapide (voire plus), et beaucoup plus portable que les clés externes.

Dans un futur proche, nous allons implémenter `FOREIGN KEY` de manière à sauver les informations dans la table de spécification, pour qu'elles soient accessibles par `mysqldump` et ODBC.

5.3.5.1 De bonnes raisons de ne pas utiliser les clés étrangères

Il y a tant de problèmes avec `FOREIGN KEY`s qu'on ne sait même pas par ou commencer :

- Les clés externes compliquent la vie, car les définitions des clés externes doivent être enregistrées dans une base de données, et les implémenter nous forcera à abandonner la gestion actuelle des tables (une table égale un fichier, qui peut être sauvé ou copié).
- L'impact sur la vitesse d'exécution de `INSERT` et `UPDATE`, et dans ce cas, presque toutes les vérifications dues à `FOREIGN KEY` ne servent à rien, car vous insérez les informations dans le bon ordre, naturellement.
- Les clés externes imposent l'utilisation de verrous sur de nombreuses tables, et cela peut conduire, par effet de domino, à geler toute la base. Il est beaucoup plus rapide d'effacer un enregistrement d'une table, puis de s'occuper des autres tables à la suite.
- Avec les clés externes, il n'est plus possible de reconstituer une table par un effacement complet, suivi d'un chargement de tous les enregistrements (depuis une nouvelle source, ou une sauvegarde).
- Avec les clés externes, il est impossible faire un dump d'une table, puis de la recharger, à moins de le faire dans un ordre très spécifique.
- Il est très facile de créer des références circulaires, qui sont autorisées, mais rendent impossible la recréation de la table à partir d'un enregistrement unique, même si la définition de la table fonctionne et est utilisable.

Le seul aspect intéressant de `FOREIGN KEY` est de donner aux clients ODBC et quelques autres la possibilité de voir comment une table est connectée, et de l'utiliser pour créer un diagramme de connexion lors de la création d'applications.

MySQL gérera prochainement les définitions des `FOREIGN KEY`, ce qui permettra aux clients de réclamer la structure de la connexion originale. La version courantes des fichiers ``.frm'' ne le permet pas.

5.3.6 Vues

MySQL ne supporte pas les vues, mais c'est sur la liste des fonctionnalités futures.

5.3.7 `--' comme début de commentaire

Sur d'autres bases SQL les commentaires commencent par ``--''. ***MySQL*** utilise ``#'' pour débuter un commentaire, même si `mysql` supprime aussi les lignes qui commencent par ``--''. Vous pouvez aussi utiliser le style de commentaires C `/* Ceci est un commentaire */` avec ***MySQL***. Comments.

MySQL n'accepte pas les commentaires commencant par ``--''; car ce style de commentaire obsolète a déjà causé de nombreux problèmes avec les requêtes générées automatiquement, lorsque la base utilise un code comme celui ci : la valeur de paiement va être placée à la place de `!paiement!`:

```
UPDATE nom_table SET credit=credit-!paiement!
```

Mais que ce passe t il si la valeur de `paiement` est négative?

Etant donné que `1−−1` valide en SQL, nous pensons que les commentaires commencé par ``−−'' sont une très mauvaise idée.

Si vous avez un programme SQL qui contient des commentaires avec le format ``−−'' vous devriez utiliser:

```
shell62; replace " −−" " #" 60; Fichier−texte−avec−des−commentaires−zarbi | mysql database
```

A la place de l'habituel :

```
shell62; mysql database 60; text−file−with−funny−comments.sql
```

Vous pouvez aussi utiliser la commande fichier ``in place" pour remplacer les commentaires ``−−'' par ``#'':

```
shell62; replace " −−" " #" −− text−file−with−funny−comments.sql
```

Retrouvez vos situation initiale avec :

```
shell62; replace " #" " −−" −− text−file−with−funny−comments.sql
```

5.4 Quels sont les standards que respecte MySQL?

SQL92. ODBC level 0−2.

5.5 Comment se débrouiller sans `COMMIT/ROLLBACK`

MySQL ne supporte pas `COMMIT−ROLLBACK`. Le problème tient à ce que `COMMIT−ROLLBACK` requiert une structure de table completement différente ce celle qui est utilisée actuellement. ***MySQL*** aurait besoin d'autres threads pour nettoyer automatiquement les tables et l'utilisation du disque serait augmentée d'autant. Ce qui ralentirait ***MySQL*** d'un facteur de 2 à 4. ***MySQL*** est beaucoup plus rapide que bien d'autres bases SQL (en général, de 2 à 3 fois plus rapide.). Une des raisons de cette vélocité est l'absence de `COMMIT−ROLLBACK`.

Pour le moment, nous sont concentrés sur l'implémentation du langage de serveur SQL (comme par exemple, les fonctions enregistrées). Avec ces fonctions, vous ne devriez faire appel au `COMMIT−ROLLBACK` que de manière exceptionnelle.

Les boucles que nécessite les transactions peuvent être remplacées avantageusement par les `LOCK TABLES`, et vous n'aurez pas besoin de curseurs si vous pouvez modifier des enregistrements à la volée.

Les transactions et les curseurs sont sur la liste de voeux, mais elles ne sont pas prioritaires. Si nous les implémentons, ce sera sous la forme d'une option de `CREATE TABLE`. Cela signifie que `COMMIT−ROLLBACK` ne fonctionnera qu'avec ces tables, ce qui ne pénalisera que les tables qui auront cette option, et non plus la base toute entière.

Ici, à TcX, nous avons un plus grand besoin d'une base de données rapide que d'une base de données généraliste. Si nous trouvons un moyen d'ajouter ces fonctionnalités sans perte de vitesse, nous le ferons sûrement. Pour le moment, nous avons d'autres choses plus importantes. Reportez tous à la liste des voeux pour savoir quelles sont les priorités (les clients de haut niveau de support peuvent cependant faire bouger les choses).

Le problème actuel est au niveau de ROLLBACK. Sans le ROLLBACK, vous pouvez effectuer des commandes de type COMMIT en utilisant la commande LOCK TABLES. Pour permettre le support du ROLLBACK, ***MySQL*** devra enregistrer tous les vieux enregistrements qui sont modifiés ou effacé, pour pouvoir les restituer si un ROLLBACK intervient. Pour les cas simples, ce n'est pas compliqué à mettre en place (l'utilitaire actuel isamlog peut servir à ce propos), mais cela peut se révéler plus compliqué pour les commandes ALTER/DROP/CREATE TABLE.

Pour éviter d'utiliser le ROLLBACK, vous pouvez suivre la stratégie suivante :

1. Utilisez LOCK TABLES ... pour verrouiller les tables auxquelles vous accédez
2. Testez les conditions d'utilisation.
3. Modifiez si tout est OK.
4. Utilisez UNLOCK TABLES pour libérer la table.

Généralement, cette méthode est beaucoup plus rapide que les transactions, et le ROLLBACK est souvent possible, mais pas toujours. Le seul point critique est que si le threads est tué au milieu de la modification, les verrous seront libérés, mais une partie des modifications ne sera pas faites.

Vous pouvez aussi utiliser les fonctions qui modifient un seul enregistrement à la fois. Vous pouvez créer des applications très efficaces avec la technique suivante :

- Modifier un champs par rapport à sa valeur actuelle
- Modifier seulement les champs qui ont changés

Par exemple, lors vous modifiez les informations concernant un client, ne modifiez que les informations qui ont changées, et non pas celle qui sont restées constantes. La recherche des valeurs est faites avec la clause WHERE de la commande UPDATE. Si l'enregistrement a changé, on peut retourner au client une message du type : "Les informations n'ont pas été modifiées, car un autre utilisateur est en train de modifier les valeurs ". Alors, on affiche l'ancienne valeur et la nouvelle, ce qui permet à l'utilisateur de décider quelle version utiliser.

Cela fournit un mécanisme du genre ``verrouillage de colonne'' mais c'est en fait un peut mieux, car seule les colonnes qui en on besoin sont utilisées. Une commande UPDATE ressemblera alors à ceci :

```
UPDATE tablename SET pay_back=pay_back+'différence de valeur';
UPDATE customer
  SET
    customer_date='date actuellee',
    address='nouvelle adresse',
    phone='nouveau numero de telephone',
    débit =débit + 'autre_débit'
  WHERE
    customer_id=id AND address='vieille addresse' AND phone='vieux numero de telephone';
```

Comme vous pouvez le voir, c'est une méthode très efficace, et qui fonctionne même si un autre client a changé la valeur entre temps.

Dans certains cas, l'utilisateur ont demandé le ROLLBACK et/ou LOCK TABLES dans le but de gérer des identifiants uniques dans des tables . Il vaut mieux utiliser le type de colonne AUTO_INCREMENT et la fonction SQL LAST_INSERT_ID() ou l' API C : mysql_insert_id(). mysql_insert_id().

A TcX, nous n'avons jamais eu besoin d'un verrouillage de ligne, car nous avons toujours réussit à contourner le problème. Dans certains cas, le verrouillage de ligne était nécessaire, mais c'est très rarement le cas. Si vous voulez le verrouillage de ligne, utiliser un flag sur une colonne dans la table comme ci :

```
UPDATE nom_table SET row_flag=1 WHERE id=ID;
```

MySQL retourne 1, ce qui correspond au nombre de ligne affectées par la commande, si la ligne a été trouvée, et que le flag `row_flag` n'était pas déjà à 1.

Vous pouvez considérer que ***MySQL*** traite la requête ci dessus de la manière suivante :

```
UPDATE nom_table SET row_flag=1 WHERE id=ID and row_flag 60;62; 1;
```

6 Systèmes de droits MySQL

MySQL dispose d'un système moderne mais original de droits d'accès. Cette section le décrit.

6.1 A quoi sert le système de droits

La fonction primaire du système de droits de ***MySQL*** est d'authentifier un utilisateur se connectant, et l'associer avec les droits d'utilisation des commandes ***select***, ***insert***, ***update*** et ***delete*** sur cette base.

Les fonctions secondaires inclus la possibilité d'accueillir un utilisateur anonyme, et de donner des droits particuliers à des fonctions spécifiques à ***MySQL*** `LOAD DATA INFILE` et les opérations.

6.2 Noms et mot de passe des d'utilisateurs MySQL

Il y a de grandes différences entre la gestion des noms d'utilisateur et mots de passe de ***MySQL***, et celle de Unix ou de Windows.

- Les noms d'utilisateurs, utilisé par ***MySQL*** pour l'authentification, n'ont rien à voir avec les nom d'utilisateur de Unix (Nom de login) ou de Windows. La plus part des clients ***MySQL*** essaie de se connecter à la base avec le nom d'utilisateur Unix courant, mais cela est uniquement par commodité. Les programmes clients permettent de se connecter sous un autre nom, spécifié avec l'option `-u` ou `--user`. Cela signifie que vous ne pouvez sécuriser un serveur ***MySQL*** qu'en ayant un mot de passe pour chacun des utilisateurs. Sinon, n'importe quelle personne qui se connecte en utilisant n'importe quel nom, réussira à se connecter, si ce nom d'utilisateur n'a pas de mot de passe.
- ***Les noms d'utilisateurs MySQL*** peuvent avoir jusqu'à 16 caractères de long. Généralement, les noms d'utilisateur Unix sont limités à 8 caractères.
- ***Les mots de passe MySQL*** n'ont rien à voir non plus avec le mot de passe Unix ou Windows. Il n'y a pas de connexion entre les deux, mais ils peuvent être identiques.
- ***Les mots de passe MySQL*** sont cryptés avec un cryptage différents de celui d'Unix. Reportez vous à la description de la commande `PASSWORD()et ENCRYPT()` dans la section 7.3.12 Fonctions diverses.

6.3 Connection au serveur MySQL

Les clients ***MySQL*** ont besoin d'un certains nombre de paramètres pour se connecter à un serveur ***MySQL*** : l'hôte qui abrite le serveur, le nom d'utilisateur et le mot de passe. Par exemple, le client `mysql` peut être lancé avec la ligne suivante : (les arguments optionnels sont mis entre crochets ``['' et ``]'') :

```
shell62; mysql [-h host_name] [-u user_name] [-pyour_pass]
```

Il est aussi possible de remplacer respectivement les options `-h`, `-u` et `-p` par`--host=host_name`, `--user=user_name` et `--password=your_pass`. Notez bien qu'il n'y as pas d'espace entre `-p` ou `--password=` et le mot de passe qui le suit.

Note: Transmettre un mot de passe dans la ligne de commande n'est pas sécurisé ! N'importe quel utilisateur du système peut découvrir le mot de passe en tapant : `ps auxww`. 4.15.4 Fichier d'options.

Par défaut, mysql utilise les valeurs suivantes :

- Le nom d'hôte par défaut est `localhost`, c'est à dire la machine locale.
- Le nom d'utilisateur par défaut est le nom de login Unix.
- Aucun mot de passe n'est envoyé si `-p` n'est pas précisé.

Ainsi, pour un utilisateur Unix `joe`, les commandes suivantes sont équivalentes :

```
shell62; mysql −h localhost −u joe
shell62; mysql −h localhost
shell62; mysql −u joe
shell62; mysql
```

D'autres clients ***MySQL*** se comportent de manière similaire :

Sous Unix, vous pouvez spécifier d'autres valeurs par défaut, lors de la connexion, ce qui vous évitera d'entrer les paramètres à chaque connexion. Cela peut être fait de diverses façons :

- Vous pouvez spécifier les paramètres de connexion dans la section `[client]` du fichier de configuration ``.my.cnf'' de votre dossier personnel. Les informations doivent être présentées comme suit :

```
[client]
host=host_name
user=user_name
password=your_pass
```

4.15.4 Fichier d'options

- Vous pouvez spécifier les paramètres de connexion en utilisant les variables d'environnement. L'hôte sera spécifié avec la variable `MYSQL_HOST`. Le nom d'utilisateur ***MySQL*** peut être spécifié avec les variables `USER`, `LOGNAME` ou `LOGIN` (bien que ces variables peuvent avoir déjà été réservé pour votre login de connexion, et ce ne serait pas conseillé de les changer). Le mot de passe sera spécifié dans la variable `MYSQL_PWD` (mais ce n'est pas très sur, comme vous le verrez à la prochaine section).

Si les paramètres de connexion sont spécifiables de nombreuses manières, les valeurs spécifiées sur la ligne de commande ont priorité sur les valeurs enregistrées dans le fichier de configuration, et ces dernières ont priorité sur les variables d'environnement.

6.4 Un mot de passe sûr

Il n'est pas conseillé d'exposer son mot de passe, ou de le stocker dans des endroits qui faciliterait sa connaissances par d'autres utilisateurs. Les méthodes de spécifications du mot de passe lors du lancement du programme client sont présentées ci−dessous, avec à chaque fois, les risques de la méthode :

- Utiliser la ligne de commande avec l'option `−pyour_pass` ou `−−password=your_pass`. C'est très pratique, mais très dangereux, ca le mot de passe devient visible pour les commandes de statut du système (comme par exemple `ps`) qui peut être invoqué pas d'autres utilisateurs. Les clients ***MySQL*** remplace le mot de passe par une ligne de zéro lors de la séquence d'initialisation, mais il y a quand même un temps très bref, où la valeur est écrite en clair).
- Utiliser la ligne de commande avec l'option `−p` ou `−−password` (sans valeur spécifiée). Dans ce cas, le programme client va solliciter le mot de passe depuis la console :

```
shell62; mysql −u user_name −p
Enter password: ********
```

Le client renvoie les caractères ``*'' à la console, au fur et à mesure que vous entrez votre mot de passe, ce qui fait que quelqu'un derrière votre épaule ne pourra pas le voir. La sécurité est donc renforcée par rapport à la méthode précédente, mais cela oblige à utiliser le client en mode interactif. Vous ne pourrez pas lancer le programme depuis un script.

- Enregistrez votre mot de passe dans un fichier de configuration. Par exemple, vous pouvez mettre votre mot de passe dans la section `[client]` du fichier de configuration ``.my.cnf'' de votre dossier home.

```
[client]
password=your_pass
```

Si vous stockez votre mot de passe dans le fichier ``.my.cnf'', ce fichier ne doit pas être accessible en

lecture ou en écriture ni au groupe, ni au monde. Assurez vous que le mode d'accès du fichier est `400` ou `600`. 4.15.4 Fichier d'options.

- Vous pouvez enregistrer votre mot de passe dans la variable d'environnement `MYSQL_PWD`, mais cette méthode doit être considéré comme extrêmement dangereuse. Certaines options de `ps` permettent d'avoir accès aux variables d'environnement des processus en cours, ce qui fait que votre mot de passe peut être parfaitement lisible. Même sur des systèmes qui n'ont pas cette version de `ps`, il est fortement déconseillé de supposer qu'il n'y a pas d'autres méthode d'observation des processus.

L'un dans l'autre, la méthode la plus sûre est de fournir le mot de passe à la console, ou de spécifier le mot de passe dans le fichier ``.my.cnf'', correctement protégé.

6.5 Droits sous MySQL

Les droits sont conservés dans les tables `user`, `db`, `host`, `tables_priv` et `columns_priv` de la base de données. Le serveur ***MySQL*** lis le contenu de ces tables au démarrage, et à chaque fois que c'est nécessaire, comme décrit dans la section 6.9 Prise en compte des modifications de droits.

Les noms utilisés dans ce manuel pour se référer aux droits de MySQL sont listés ci−dessous, avec le nom de la colonne associé à chaque droit dans la table de droits, et le contexte d'application du droit.

Les droits ***select***, ***insert***, ***update*** et ***delete*** permettent d'exécuter des opérations sur les lignes dans les tables existantes d'une base. Vous pouvez même exécuter certaines commandes `SELECT` sans avoir accès à aucune base sur le serveur. Par exemple, vous pouvez utiliser `mysql` comme une simple calculatrice :

```
mysql62; SELECT 1+1;
mysql62; SELECT PI()*2;
```

Le droit ***index*** vous permet de créer ou de détruire des index.

Le droit ***alter*** vous permet d'utiliser la commande `ALTER TABLE`.

Les droits ***create*** et ***drop*** vous permet de créer et détruire de nouvelles tables et bases.

Il faut bien remarquer que si vous donnez le droit de ***drop*** pour la base `mysql` , un utilisateur pourra effacer la table des droits ***MySQL***!

Le droit ***grant*** vous permet de donner des droits que vous possédez à un autre utilisateur.

Le droit ***file*** vous permet de lire et écrire des fichiers sur le serveur, en utilisant les commandes `LOAD DATA INFILE` et `SELECT ... INTO OUTFILE`. Tout utilisateur qui dispose de ce droit peut écrire ou lire des fichiers que ***MySQL*** peut écrire ou lire.

Les autres droits sont utilisés pour les opérations administratives, qui sont disponibles avec `mysqladmin`. La table ci−dessous montre quelle droit donne accès à quelle commande `mysqladmi`:

La commande `reload` force le serveur à relire les tables de droits. La commande `refresh` vide toutes les tables de la mémoire, et ouvre puis ferme les fichiers d'historique. `flush−privileges` est un synonyme de `reload`. Les autres commandes `flush−*` exécutent des fonctions similaires à `refresh` mais ne sont pas aussi limitées, et parfois même, elles sont préférables. Par exemple, si vous souhaitez uniquement enregistrer les fichiers d'historique, `flush−logs` est mieux que `refresh`.

La commande `shutdown` éteint le serveur.

La commande `processlist` affiche la liste des threads courant du serveur. La commande `kill` arrête les threads du serveur. Vous pouvez toujours afficher et détruire les threads que vous possédez, mais vous devez avoir les droits de ***process*** pour afficher ou terminer les threads d'un autre utilisateur.

La meilleure stratégie est de ne donner des droits qu'à ceux qui en ont besoin, mais il faut être particulièrement prudents avec certains droits :

- Le droit de ***grant*** permet à utilisateur de donner ses propres droits à un autre. Deux utilisateurs pourront alors s'échanger les droits, et ainsi, obtenir des droits supplémentaires.
- Les droits de ***alter*** peuvent être utilisés pour renommer des tables systèmes
- Les droits de ***file*** peuvent être détourné : en lisant n'importe quel fichier (sensible de préférence) sur le serveur pour le monter dans une base de données, il est alors possible d'accéder au contenu en utilisant la commande `SELECT`.
- Les droits de ***shutdown*** peuvent être utilisé pour empêcher l'accès au serveur, en l'éteignant.
- Les droits de ***process*** permettent à un utilisateur de voir les commandes des autres utilisateurs, sans cryptage, notamment les commandes de changement de mot de passe.
- Les droits sur la base `mysql` peuvent être utilisés pour changer des mots de passe, ou tout autre droit d'accès. (Les mots de passes sont cryptés avant d'être enregistrés, ce qui empêche leur relecture. Mais avec les droits adéquats, un utilisateur peu scrupuleux peu les remplacer par d'autre mot de passe, et empêcher l'accès).

Il y a des limitations avec les droits ***MySQL***:

- Vous ne pouvez pas refuser des droits à un utilisateur. Vous ne pouvez pas spécifier un utilisateur, et lui refuser spécialement les droits de connexion.
- Vous ne pouvez pas séparer les droits de création et d'effacement des tables et de la bases de données qui les contient. L'utilisateur pourra créer et effacer les tables ET la base.

6.6 Fonctionnement du système de droits

MySQL s'assure que tous les utilisateurs peuvent faire ce qu'ils ont le droit de faire. Lorsque vous vous connectez à un serveur ***MySQL***, le serveur détermine votre identité gr ce à ***l'hôte depuis lequel vous vous connectez,*** et ***le nom d'utilisateur que vous spécifiez.***. Le système vous alloue alors les droits adéquats.

MySQL considère que le nom de l'hôte et le nom d'utilisateur sont suffisants pour vous identifier sans ambiguïté, car il y a peu de chance qu'un nom d'utilisateur soit utilisé par la même personne, depuis tous les hôtes sur Internet ! Par exemple, l'utilisateur `bill` qui se connecte depuis `whitehouse.gov` ne sera probablement pas la même personne que l'utilisateur `bill` qui se connecte depuis `microsoft.com`. ***MySQL*** vous permet de distinguer les deux utilisateurs, et de donner des droits différents pour le même nom d'utilisateur, mais pour des hôtes différents.

MySQL contrôle l'accès en deux temps :

- Etape 1: le serveur vérifie que vous êtes autorisés à vous connecter.
- Etape 2: Si vous pouvez vous connecter, le serveur vérifie chaque requête que vous émettez pour voir sir vous avez des droits suffisants. Par exemple, pour sélectionner des lignes dans une table, ou effacer une table dans une base, le serveur s'assure que vous avez les droits de ***select*** ou de ***drop*** pour la base de données courante.

Le serveur utilise les tables `user`, `db` et `host` de la base `mysql` pour conserver les informations de connexion et les droits. Les champs de ces tables sont les suivants :

Pour la deuxième étape d'accès, le serveur peut consulter éventuellement (si la requête implique des tables)les tables `tables_priv` et `columns_priv`. Les champs de ces tables sont les suivants :

Chaque table de droits contient les champs d'identification et de droits.

Les champs d'identification détermine le contexte d'application des droits. Par exemple, la table `user` avec une ligne dont les champs `Host, User` ont les valeurs de `'thomas.loc.gov'` et `'bob'` seront

utilisés pour identifier les connexions de `bob` depuis `thomas.loc.gov`. De la même façon, la table `db` avec une ligne dont les champs `Host`, `User` et `Db` sont respectivement `'thomas.loc.gov'`, `'bob'` et `'reports'` seront utilisés lors que `bob` se connecte depuis host `thomas.loc.gov` pour accéder à la base `reports`. Les tables `tables_priv` et `columns_priv` contiennent les noms des tables et des tables / colonnes pour qui s'appliquent les droits.

Pour des raisons de vérifications d'accès, les comparaisons effectuées sur les colonnes `Host` sont insensibles à la casse.. `User`, `Password`, `Db` et `Table_name` sont sensibles à la casse.. `Column_name` est insensible à la casse depuis ***MySQL*** 3.22.12.

Les champs de droits indiques quels droits sont donnés, quelles commandes sont autorisées. Le serveur combine les informations des différentes tables pour construire une fiche complète de description des droits du serveur. Les règles utilisées pour cela sont décrites dans la section 6.8 Contrôle d'accès, étape 2 : vérification des requêtes.

Les champs d'identification sont des chaînes, déclarées comme ci−dessous. La valeur par défaut de chaque champs est la chaîne vide.

Dans les tables `user`, `db` et `host`, tous les champs de droits sont déclarés comme des `ENUM('N','Y')` − chaque champs peut prendre la valeur `'N'` ou `'Y'`, et la valeur par défaut est `'N'`.

Dans les tables `tables_priv` et `columns_priv`, Les champs de droits sont déclarés comme des `SET`:

Présenté rapidement, le serveur utilises les tables de droits comme ceci :

- La table `user` détermine le droit de connexion. Pour les connexions autorisées, les droits globaux de l'utilisateurs sont précisés.
- Les tables `db` et `host` sont utilisées ensembles :
  - La table `db` détermine quelles bases sont accessibles à quels utilisateurs. Les champs de droits déterminent quels sont les droits autorisés.
  - La table `host` est utilisée comme une extension de `db` lorsque vous voulez qu'une ligne de `db` s'applique à plusieurs hôtes. Par exemple, si vous voulez qu'un utilisateur soit capable d'accéder à la base depuis plusieurs hôtes différents, laissez le champs `Host` de la table `db` vide, puis ajoutez un enregistrement dans la table `host` pour chaque hôte à autoriser. Ce mécanisme est décrit en détails dans la section 6.8 Contrôle d'accès, étape 2 : vérification des requêtes.
- Les tables `tables_priv` et `columns_priv` sont similaires à la table `db`, mais s'appliquent au niveau table et colonne, plutôt qu'au niveau base.

Notez bien que les droits administratifs (tels ***reload***, ***shutdown***, etc.) ne sont spécifiés que dans la tables `user`. En effet, les opérations administratives sont des opérations sur le serveur lui−même, et ne sont pas spécifiques à une base de données : il n'y a pas de raison d'avoir ces privilèges dans d'autres tables. En fait, seule la table `user` est consultée pour déterminer les droits administratifs.

Le droit ***file*** est spécifié seulement dans la table `user`. Ce n'est pas un droit administratif, mais il n'est pas dépendant de la table, ou de la base en cours.

Le serveur `mysqld` lit le contenu des tables de droits au démarrage. Les modifications des tables de droits ne prennent effets qu'à des moments précis, comme précisé dans la 6.9 Prise en compte des modifications de droits.

Lorsque vous modifiez le contenu des tables de droits, il es important de s'assurer que les modifications de droits produiront bien l'effet désiré. Il existe une aide pour comprendre ces erreurs, reportez vous à la section `Security`.

Un outil d'analyse pratique est le script `mysqlaccess`, qui est fournit gracieusement par Yves Carlier dans la distribution de ***MySQL***. Lancez `mysqlaccess` avec l'option `−−help` pour comprendre comment il

fonctionne. Notez que `mysqlaccess` utilise les droits des tables `user`, `db` et `host`. Il ne prend pas en compte les droits sur les tables ou les colonnes.

6.7 Contrôle d'accès, étape 1 : vérification de la connexion

Lorsque vous tentez de vous connecte à un serveur ***MySQL***, le serveur accepte ou rejete votre tentative en fonction de votre identité, et de votre capacité à fournir le mot de passe correct. Dans le cas contraire, le serveur refuse complètement l'accès. Sinon, le serveur accepte la connexion, et passe en niveau 2, pour attendre les requêtes.

L'identité lors de la connexion est basé sur deux éléments :

- L'hôte depuis lequel vous vous connectez
- Votre nom d'utilisateur ***MySQL***

La vérification de l'identité est faite en utilisant les trois champs d'identité de la table `user` (`Host`, `User` et `Password`). Le serveur n'acceptera une connexion que si il existe un enregistrement qui contienne votre hôte, votre nom d'utilisateur et votre mot de passe.

Les valeurs de la table `user` peuvent être fournies avec les propriétés suivante :

- `Host` peut être un nom d'hôte, ou une adresse IP ou `'localhost'` pour indiquer la machine locale.
- Vous pouvez utilisez les caractères spéciaux ``'%'' et ``'_'' dans le champs `Host`.
- `Host` qui vaut `'%'` accepte tous les hôtes. `Host` vide est équivalent à to `'%'`. Notez que ces valeurs accepte n'importe quel hôte qui peut créer une connexion avec votre serveur !
- Les caractères spéciaux ne sont pas autorisés dans le champs `User`, mais vous pouvez le laisser vide, ce qui équivaudra à `'%'`. Si un enregistrement de la table `user` a un champs `User` vide, cet utilisateur sera considéré comme anonyme (utilisateur sans nom) plutôt qu'étant l'utilisateur avec le nom spécifié par le client. Cela signifie que l'utilisateur vide sera utilisé ultérieurement pour toutes les vérifications de droits, durant toute la connexion.
- Le champs `Password` peut être laissé vide. Cela ne signifie pas que n'importe quel mot de passe sera accepté, mais que l'utilisateur doit se connecter SANS mot de passe.

Les valeurs de `Password` qui ne sont pas vides représentent des mots de passe cryptés. ***MySQL*** ne conserve pas les mots de passe en clair, qui risqueraient ainsi d'être vu par n'importe qui. Lors de la connexion, le mot de passe fourni est crypté de la même manière que le mot de passe enregistré, et les deux valeurs sont comparées. Si les deux correspondent, le mot de passe fourni est le bon.

Les exemples suivants montrent différentes combinaisons des colonnes `Host` et `User` de la table `user` :

Etant donné que l'on peut utiliser des jokers dans le champs `Host` (i.e., `'144.155.166.%'` pour accepter tout un sous domaine), il est possible que quelqu'un essaie d'exploiter ceci en prenant comme nom `144.155.166.petaouschnock.com`. Pour contrer ce genre de tentatives, ***MySQL*** n'accepte pas les noms d'hôtes qui commencent par des chiffres suivis d'un point. Ainsi, un hote du genre `1.2.foo.com` ne pourra jamais être accepté. Seule les IP numériques peuvent utiliser un joker.

Une connexion entrante peut disposer de plusieurs entrées dans la table `user`. Par exemple, la connexion `thomas.loc.gov` de `fred` pouurait être acceptée plusieurs fois dans les exemples ci−dessus. Comment le serveur fait il pour choisir une entrée ou une autre ? Il résoud la question en triant les utilisateurs au moment du démarrage, et en lisant les entrées dans cet ordre trié. La première ligne qu'il trouve est alors la bonne.

La table `user` fonctionne comme ceci. Supposons qu'elle ressemble à ceci :

```
+−−−−−−−−−−+−−−−−−−−−−+−
| Host      | User     | ...
```

```
+-----------+----------+-
| %         | root     | ...
| %         | jeffrey  | ...
| localhost | root     | ...
| localhost |          | ...
+-----------+----------+-
```

Lorsque le serveur lit la table, il classe les lignes en commencant par les hôtes les moins généraux (`'%'` dans la colonne `Host` signifie ``tous les hôtes'' et c'est le cas le plus général). Les lignes avec le même hôte sont classé en commencant par les utilisateurs les moins généraux (un utilisateur en blanc signifie ``tous les utilisateurs'' et c'est le cas le plus général. Une fois triée, la table ressemble à ceci :

```
+-----------+----------+-
| Host      | User     | ...
+-----------+----------+-
| localhost | root     | ...
| localhost |          | ...
| %         | jeffrey  | ...
| %         | root     | ...
+-----------+----------+-
```

Lors d'une connexion, le serveur recherche parmi les lignes classées, et utilise la première ligne qui aille et l'utilise. Si la connexion entrante vient de l'hôte `localhost` et de l'utilisateur `jeffrey`, les premières lignes, avec `'localhost'` sont trouvées. Parmi celles−ci, la ligne avec le champs utilisateur vide correspond à la connexion. (la linge avec `'%'/'jeffrey'` était aussi correcte mais elle apparaît plus tard dans la table).

Voici un autre exemple :. fonctionne comme ceci. Supposons que la table `user` ressemble à ceci :

```
+----------------+----------+-
| Host           | User     | ...
+----------------+----------+-
| %              | jeffrey  | ...
| thomas.loc.gov |          | ...
+----------------+----------+-
```

Une fois triée, la table ressemble à ceci :

```
+----------------+----------+-
| Host           | User     | ...
+----------------+----------+-
| thomas.loc.gov |          | ...
| %              | jeffrey  | ...
+----------------+----------+-
```

Une connexion par l'hôte `thomas.loc.gov` et l'utilisateur `jeffrey` utilisera la première ligne, et la même connexion depuis `whitehouse.gov` utilisera la seconde.

Une erreur commune est de penser que pour un nom d'utilisateur donné, toutes les lignes qui nomme explicitement cet utilisateur seront utilisées en premier par le serveur pour accepter la connexion. Ceci est tout simplement faux. L'exemple précédent l'illustre bien : la connexion depuis `thomas.loc.gov` par `jeffrey` n'utilise par la ligne contenant l'utilisateur `'jeffrey'` mais la ligne sans nom d'utilisateur.

Si vous avez des problèmes à vous connecte au serveur, afficher la table `user` et trier la à la main, pour

savoir quelle ligne est la première a accepter votre connexion.

6.8 Contrôle d'accès, étape 2 : vérification des requêtes

Une fois que la connexion est établie, le serveur passe en niveau 2. Pour chaque requête entrante, le serveur va vérifier que les droits sont suffisants pour effectuer la requête, en fonction du type d'opération. C'est ici qu'interviennent les champs de droits des tables de droits. Ces droits peuvent être stockés dans les tables `user`, `db`, `host`, `tables_priv` ou `columns_priv` tables. Ces tables sont gérées gr ce aux commandes `GRANT` et `REVOKE`. Reportez vous à la section `Privileges`, dans laquelle les champs de ces tables sont présentés).

Les droits pris en charge par la table `user` sont assignés globalement, et s'appliquent quelque soit la base de données courante. Par exemple, si la table `user` donne les droits de ***delete***, vous pouvez efface n'importe quelle base de donnée sur le serveur ! ! En d'autres termes, la table `user` fourni les droits de super utilisateur, et il est avisé de ne donner ces droits qu'aux administrateurs du serveur. Pour les autres utilisateurs, il vaut mieux laisser les droits à `'N'` et donner des droits spécifiques sur les bases de données, avec les tables `db` et `host`.

Les tables `db` et `host` donnent des droits spécifiques aux bases de données. Les valeurs acceptées dans les champs sont les suivantes :

- Les caractères spéciaux ``%'' et ``_'' peuvent être utilisés dans les champs `Host` et `Db` de deux tables.
- Un `'%'` dans le champs `Host` de la table `db` signifie ``tous les hôtes." Une valeur vide pour `Host` dans la table `db` signifie ``consulte la table `host` pour plus de détails."
- Un `'%'` ou une chaîne vide dans le champs `Host` de la table `host` signifie `` tous les hôtes."
- Un `'%'` ou une chaîne vide dans le champs `Db` dans l'une des tables signifie ``toutes les bases de données."
- une chaîne vide dans le champs `User` dans l'une des tables corresponde à l'utilisateur anonyme.

Les tables `db` et `host` sont lues et triées au démarrage du serveur (en même temps que la table `user`). La table `db` est triée en fonction des champs `Host`, `Db` et `User`, et la table `host` table est triée en fonction des champs `Host Db`. Tout comme pour la table `user`, le trie met les valeurs les plus précises en premier, puis le plus larges. Lors de l'utilisation de la table, la première valeur qui correspond à l'utilisateur est utilisée.

Les tables de droits `tables_priv` et `columns_priv` contiennent les droits spécifiques aux droits par table et par colonne. Les valeurs des champs peuvent être les suivantes :

- Les caractères spéciaux ``%'' et ``_'' peuvent être utilisés dans les champs `Host` et `Db` de deux tables.
- Un `'%'` ou une chaîne vide dans le `Host` de la table `db` signifie ``tous les hôtes."
- Les champs `Db`, `Table_name` et `Column_name` ne peuvent pas contenir de champs vide, ou de caractères spéciaux.

Les tables `tables_priv` et `columns_priv` sont triées en fonction des champs `Host`, `Db` et `User`. Le tri est identique à celui de la table `db`, mais étant donné que le champs `Host` ne contient pas de caractère spéciaux, le tri est nettement plus simple.

Le processus de vérification de requête est décrit ci−dessous. (Si vous êtes familier avec la programmation de contrôle d'accès, vous remarquerez que la description qui suit est un peu simplifiée, par rapport aux algorithmes utilisés. En fait, la description est équivalente au code, et la différence sert simplement à rendre la description plus accessible).

Pour les requêtes administratives telles que ***shutdown***, ***reload***, etc..., le serveur ne vérifie les droits que dans la table `user`, étant donné que c'est la seule qui spécifie les droits administratifs. La commande est exécutée si les droits sont disponibles, et sinon, la requête n'est pas autorisée. Par exemple, si vous voulez exécuter `mysqladmin shutdown` mais que votre compte utilisateur dans la table `user` n'a pas les droits de

shutdown, l'autorisation n'est aps donnée, sans même vérifier les tables `db` ou `host`. (Etant donné que ces tables ne contiennent pas de colonne `Shutdown_priv`, il n'y a pas besoin de passer en revue ces tables.)

Pour les requêtes liées aux bases de données, telles que ***insert***, ***update***, etc., le serveur commence par vérifier les droits globaux (droits de super utilisateur) en recherchant dans la table `user`. Si il trouve des droits, l'exécution de la requête est autorisé. Si les droits globaux sont insuffisants, le serveur détermine les droits spécifiques à cette base en vérifiant les tables `db` et `host`:

1. Le serveur dans la table `db` une ligne qui corresponde à `Host`, `Db` et `User` de l'utilisateur. `Host` et `User` ont été défini lors de la connexion au serveur. Le champs `Db` prendre le nom de la base de données qui va être modifiée. Si il n'y a aucune entrée, l'autorisation n'est pas donnée.
2. Si il y a une ligne, et que le champs `Host` n'est pas laissé vide, cette ligne définit les droits de l'utilisateur, spécifiques à cette base.
3. Si le champs `Host` a été laissé vide, cela signifie que la table `host` contient la liste des hôtes qui ont l'autorisation d'accéder à cette base. Dans ce cas, une nouvelle recherche est effectuée dans la table `host` pour rechercher une ligne qui correspondent à `Host` et `Db`. Si aucune ligne n'est trouvée, alors l'accès n'est pas autorisé. Si une ligne est trouvée, les droits d'accès de cet utilisateur sont représentés par l'intersection des droits issus de la table `db` et de la table `host`, i.e., c'est à dire les droits qui sont à `'Y'` dans les deux lignes trouvées. (De cette manière, vous pouvez donner des droits généraux, et restreindre sélectivement en fonction des hôtes, gr ce à la table `host`.)

Après avoir déterminé les droits spécifiques à la base de données, avec les tables `db` et `host`, le serveur les ajoutent aux droits globaux donnés par la table `user`. Si le résultat de cette union autorise l'opération, la requête est exécutée. Sinon, le serveur vérifie les droits sur les tables et les colonnes dans les tables `tables_priv` et `columns_priv` et les ajoutent aux droits de l'utilisateur. l'accès est alors donné ou retiré en fonction du résultat.

Exprimé par une formule booléenne, la description précédente du calcul des droits est la suivante :

```
global privileges
OR (database privileges AND host privileges)
OR table privileges
OR column privileges
```

Il n'est pas évident que si les droits globaux `user` sont insuffisants pour l'opération demandée, le serveur va les ajouter dans les différentes tables qui conservent les droits. La raison est qu'une requête peut nécessiter plusieurs droits différents. Par exemple, la commande `INSERT ... SELECT` requiert les droits d'insertion (***insert)*** et de selection (***select)***. Il se peut alors que les droits d'insertion soient disponible au niveau de la table, et que les droits de selection soient au niveau de la colonne. Dans ce cas, les droits de deux niveaux doivent être combinés pour autoriser la commande. D'où cette propagation de droits.

La table `host` est utilisée pour avoir une liste de serveurs ``sécurisés''. A TcX, la table `host` contenait la liste de toutes les machines du réseau local. Toutes ces machines avaient des autorisations d'accès sur le serveur.

Vous pouvez aussi utiliser cette table pour lister les serveurs qui ne sont pas sûrs. Par exemple, supposons que la machine `public.votre.domaine` soit située dans une zone publique qui ne soient pas sûre. Vous pouvez alors accepter tous les hôtes du réseau, et exclure cette machine, comme ceci :

```
+----------------------+----+−
| Host                 | Db | ...
+----------------------+----+−
| public.votre.domaine | %  | ... (tous les droits à 'N')
| %.your.domain        | %  | ... (tous les droits à 'Y')
+------------------**−+----+−
```

Bien entendu, il est plus sage de tester toutes les lignes des tables de droits (e.g., en utilisant `mysqlaccess`) pour s'assurer que les droits sont bien donnés comme vous le souhaitez.

6.9 Prise en compte des modifications de droits

Lorsque `mysqld` démarre, les tables de droits sont lu, et chargés en mémoire : ils sont alors effectifs.

Les modifications des tables de droits qui se font avec les requêtes `GRANT`, `REVOKE`, ou `SET PASSWORD` sont immédiatement prises en compte.

Si vous modifiez les tables de droits manuellement (`INSERT`, `UPDATE`, etc.), vous devrez exécuter la commande `FLUSH PRIVILEGES` ou lancer `mysqladmin flush−privileges` pour indiquer au serveur qu'il faut qu'il relise les tables de droits. Sinon, les changements ne seront pas effectifs avant le prochain rechargement.

Lorsque le serveur remarque que les tables de droits ont été changées, les connexions courantes sont traitées comme suit :

- Les droits de table et de colonnes sont pris en compte à la prochaine requête.
- Les droits de base sont pris en compte à la prochaine utilisation de la requête `USE nom_base_de_donnees`.

Les droits globaux et les mots de passes ne changent qu'à la prochaine connexion.

6.10 Droits initiaux

Après avoir installé ***MySQL***, vous allez installer les premiers droits en exécutant l'utilitaire `scripts/mysql_install_db`. 4.7.1 Introduction à l'installation rapide. Le script `scripts/mysql_install_db` démarre le serveur `mysqld`, puis initialise les tables de droits, qui contiendront alors les droits suivants :

- ***L'utilisateur MySQL*** `root` est créé, c'est le super utilisateur, investi de tous les droits. Les connexions au serveur MySQL doivent être faites depuis l'hôte local. ***NB:*** Le super utilisateur initial a un mot de passe vide ce qui fait que n'importe qui peut se connecter sans mot de passe, et disposer de tous les droits.
- Un utilisateur anonyme est aussi créé, qui peut travailler librement dans les bases de données dont le nom commence par `'test_'` et aussi `'test'` . Les connections doivent être faites depuis l'hôte local. Cela signifie que quiconque se connecte depuis cet hôte peut être traité comme un utilisateur anonyme.
- Aucun autre droit n'est attribué. Par exemple, les utilisateurs normaux ne peuvent pas utiliser `mysqladmin shutdown` ou `mysqladmin processlist`.

NB: Les droits par défaut sont différents sous Windows. 4.12.4 Faire tourner ***MySQL*** sous Win32.

Etant donné que votre installation initialle est très ouverte, la première chose à faire est d'attribuer un mot de passe au root. Vous pouvez le faire simplement avec la commande suivante :

```
shell62; mysql −u root mysql
mysql62; UPDATE user SET Password=PASSWORD(nouveau_mot_de_passe)
           WHERE user='root';
mysql62; FLUSH PRIVILEGES;
```

Vous pouvez aussi utiliser la commande `SET PASSWORD`:

```
shell62; mysql −u root mysql
mysql62; SET PASSWORD FOR root=PASSWORD(nouveau_mot_de_passe);
```

Un autre moyen d'attribuer le mot de passe est de passer la commande `mysqladmin`:

```
shell62; mysqladmin −u root password nouveau_mot_de_passe
```

Notez bien que si vous modifiez un mot de passe dans la table `user` directement avec la première méthode, vous devez faire relire les tables par le serveur avec la commande `FLUSH PRIVILEGES` : dans le cas contraire, les modifications ne seront pas prises en compte.

Une fois que le mot de passe du `root` a été affecté, vous devrez le fournir pour pouvoir vous connecter comme `root`.

Vous pouvez aussi laisser le mot de passe du `root` blanc, ce qui vous évitera d'avoir à le spécifier lors de la connexion, surtout si vous faites d'autres tests d'installation. Sinon, n'oubliez pas d'en assigner un lors du passege en production, pour ne pas créer des trous de sécurité.

Reportez vous au script `scripts/mysql_install_db` pour voir fonctionner les privilèges par défaut. Vous pouvez utiliser ce script comme base, pour ajouter de nouveaux utilisateurs.

Si vous voulez que les privilèges par défaut soient différents de ceux présentés jusqu'à présent, modifiez le script `mysql_install_db` avant de l'exécuter.

Pour recréer les tables de droits, effacez les fichiers ``*.frm'', ``*.ISM'' et ``*.ISD'' dans le dossier contenant le serveur `mysql`. (ce dossier est nommé ``mysql'' dans le dossier du serveur. Ce dernier est affiché si avec la commande `mysqld −help.` Alors, exécutez le script `mysql_install_db`, éventuellement après avoir l'avoir édité.

NOTE IMPORTANTE : pour les versions de ***MySQL*** antérieures à 3.22.10, vous ne DEVEZ PAS effacer les fichiers ``*.frm''. Si vous l'avez fait par accident, vous devez remettre une copie (tirée de la distribution ***MySQL***) avant d'exécuter `mysql_install_db`.

6.11 Ajout d'un nouvel utilisateur MySQL

Vous pouvez ajouter des utilisateurs de deux manières différentes : en utilisant la commande `GRANT` ou en manipulant directement les tables de droits ***MySQL***. La meilleure méthode est l'utilisation de la commande `GRANT`, car elle est plus concise et est la source de moins d'erreur. Les exemples ci dessous montrent comment utiliser le client `mysql` client pour ajouter un nouvel utilisateur. Ces exemples supposent que les droits ont été créé comme décrit dans la section précédente. Cela signifie notamment que pour faire des modifications, il vous faut être sur la machine qui fait tourner le serveur, vous devez vous connecter en tant que `root`, et le `root` doit avoir les droits d'insertion et de rechargement (***insert*** et ***reload***). De plus, si vous avez changé le mot de passe du `root`, vous devrez le spécifier pour pouvoir exécuter les commandes suivantes

Ajout d'un nouvel utilisateur

```
shell62; mysql −−user=root mysql
mysql62; GRANT ALL PRIVILEGES ON *.* TO monty@localhost
           IDENTIFIED BY 'quelquechose' WITH GRANT OPTION;
mysql62; GRANT ALL PRIVILEGES ON *.* TO monty@"%"
           IDENTIFIED BY 'something' WITH GRANT OPTION;
mysql62; GRANT RELOAD,PROCESS ON *.* TO admin@localhost;
mysql62; GRANT USAGE ON *.* TO dummy@localhost;
```

Ces commandes ajoutent 3 nouveaux utilisateurs

- monty Un super−utilisateur avec les pleins pouvoirs, qui peut se connecter depuis n'importe quelle machine (même distante) mais qui doit utiliser le mot de passe `'quelquechose'` pour ce faire. Notez l'utilisation de la commande `GRANT` pour les deux formes `monty@localhost` et

monty@"%". Si la ligne avec localhost n'est pas ajoutée l'utilisateur anonyme qui a été créé par le script mysql_install_db aura priorité lors de la connexion, car il sera plus spécifique. Il faut donc l'ajouter nommément pour pouvoir donner les bons droits à monty.

- admin Un utilisateur qui peut se connecter depuis localhost sans mot de passe, et qui a les droits administratifs de ***reload*** et ***process***. Cela va lui permettre d'exécuter les utilitaires mysqladmin reload, mysqladmin refresh et mysqladmin flush−*, ainsi que mysqladmin processlist . Aucun droit lié aux bases de données ne sont donnés. Ils pourront être donnés plus tard, avec des commandes GRANT.
- dummy Un utilisateur qui peut se connecter sans mot de passe, mais uniquement depuis localhost . Les privilèges globaux sont tous à 'N' − le type spécial USAGE vous permet de créer rapidement un utilisateur sans droits, et lui occtroyer plus tard.

Vous pouvez aussi ajouter ces informations avec des commandes INSERT et en forcant le serveur à recharger ces tables.

```
shell62; mysql −−user=root mysql
mysql62; INSERT INTO user VALUES('localhost','monty',PASSWORD('something'),
                'Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y')
mysql62; INSERT INTO user VALUES('%','monty',PASSWORD('something'),
                'Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y')
mysql62; INSERT INTO user SET Host='localhost',User='admin',
                 Reload_priv='Y', Process_priv='Y';
mysql62; INSERT INTO user (Host,User,Password)
                       VALUES('localhost','dummy','');
mysql62; FLUSH PRIVILEGES;
```

Suivant la version de ***MySQL***, vous pouvez avoir un nombre différent de 'Y' dans les lignes ci−dessus (Les versions antérieures à 3.22.11 avaient moins de colonnes de droits.). Pour l'utilisateur admin, la version plus pratique de la commande INSERT est disponible à partir de la version 3.22.11.

Notez que pour créer un super utilisateur, il vous suffit de créer une ligne dans la table user avec tous les privilèges mis à 'Y' : aucune ligne n'est requise dans les tables db ou host.

La colonne de droit dans la table user n'ont pas été explicitement fixée dans la dernière commande INSERT (pour l'utilisateur dummy), ce qui fait que ces colonnes ont une valeur par défaut de 'N'. C'est le même comportement que pour la commande GRANT USAGE.

L'exemple suivant ajoute un utilisateur custom qui peut se connecter depuis les hôtes localhost, server.domain et whitehouse.gov. Il veut pouvoir accéder à la base de données bankaccount, mais uniquement depuis localhost, à la base expenses mais uniquement depuis whitehouse.gov et à la base customer depuis tous les hôtes Il veut avoir le même mot de passe pour toutes les connexions.

Pour donner ces droits à cet utilisateur, utilisez les commandes GRANT suivantes :

```
shell62; mysql −−user=root mysql
mysql62; GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,DROP
           ON bankaccount.*
           TO custom@localhost
           IDENTIFIED BY 'stupid';
mysql62; GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,DROP
           ON expenses.*
           TO custom@whitehouse.gov
           IDENTIFIED BY 'stupid';
mysql62; GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,DROP
           ON customer.*
           TO custom'%'
           IDENTIFIED BY 'stupid';
```

Pour donner ces droits à cet utilisateur en accédant directement aux tables de droits, utilisez les commandes suivantes :

```
shell62; mysql −−user=root mysql
mysql62; INSERT INTO user (Host,User,Password)
       VALUES('localhost','custom',PASSWORD('stupid'));
mysql62; INSERT INTO user (Host,User,Password)
       VALUES('server.domain','custom',PASSWORD('stupid'));
mysql62; INSERT INTO user (Host,User,Password)
       VALUES('whitehouse.gov','custom',PASSWORD('stupid'));
mysql62; INSERT INTO db
       (Host,Db,User,Select_priv,Insert_priv,Update_priv,Delete_priv,
        Create_priv,Drop_priv)
       VALUES
       ('localhost','bankaccount','custom','Y','Y','Y','Y','Y','Y');
mysql62; INSERT INTO db
       (Host,Db,User,Select_priv,Insert_priv,Update_priv,Delete_priv,
        Create_priv,Drop_priv)
       VALUES
       ('whitehouse.gov','expenses','custom','Y','Y','Y','Y','Y','Y');
mysql62; INSERT INTO db
       (Host,Db,User,Select_priv,Insert_priv,Update_priv,Delete_priv,
        Create_priv,Drop_priv)
       VALUES('%','customer','custom','Y','Y','Y','Y','Y','Y');
mysql62; FLUSH PRIVILEGES;
```

Les trois premières insertions ajoutent des lignes dans la table `user` pour autoriser la connexion de `custom` depuis les divers sites qu'il désire, mais ne lui donne aucune permission. (tous ses droits sont à `'N'`). Les trois insertions suivantes ajoutent des lignes dans la table `db` pour donner des droits sur les bases `bankaccount`, `expenses` et `customer`, mais uniquement pour des connexions avec des hôtes autorisés. Et, comme toujours lorsqu'on touche aux tables de droits, le serveur doit les recharger (avec `FLUSH PRIVILEGES`) pour les prendre en compte.

Si vous désirez donner un droit spécifique à un utilisateur donnée, mais depuis n'importe quel hôte, utilisez la commande `GRANT` suivante :

```
mysql62; GRANT ...
           ON *.*
           TO myusername@"%.mydomainname.com"
           IDENTIFIED BY 'mypassword';
```

Pour faire la même chose avec une attaque directe des tables, faites ceci :

```
mysql62; INSERT INTO user VALUES ('%.mydomainname.com', 'myusername',
           PASSWORD('mypassword'),...);
mysql62; FLUSH PRIVILEGES;
```

Vous pouvez aussi utiliser `xmysqladmin`, `mysql_webadmin` et even `xmysql` pour insérer, modifier des valeurs dans les tables de droits. Vous pouvez trouvez ces utilitaires à http://www.mysql.com/Contrib/.

6.12 Comment affecter les mots de passe

Les exemples des sections précédentes illustre un principe important : lorsque vous enregistrez des mots de passe non vide avec les commandes `INSERT` ou `UPDATE`, vous devez utiliser la fonction `PASSWORD()` pour l'encrypter. En effet, la table `user` conserve les mots de passe sous forme cryptée, et non pas en clair. Si vous oubliez ceci, vous pouvez vous retrouver avec des mots de passes tels que :

```
shell62; mysql −u root mysql
```

```
mysql62; INSERT INTO user (Host,User,Password)
       VALUES('%','jeffrey','biscuit');
mysql62; FLUSH PRIVILEGES;
```

Le résultat est que la valeur en clair `'biscuit'` est enregistrée dans la table `user`. Lorsque l'utilisateur `jeffrey` va tenter de se connecter au serveur, le client `mysql` va crypter le mot de passe qui lui est fourni avec `PASSWORD()`, et l'envoyer au serveur. Le serveur compare alors la valeur qui est stockées (qui est en clair `'biscuit'`) avec la même valeur, mais encryptée (qui ne sera *pas* `'biscuit'`). La comparaison échoue, et le serveur rejette al connexion.

```
shell62; mysql −u jeffrey −pbiscuit test
Access denied
```

Etant donné que les mots de passe doivent être crypté lorsqu'ils sont inséré dans la table `user`, la commande `INSERT` doit être spécifié comme suit :

```
mysql62; INSERT INTO user (Host,User,Password)
       VALUES('%','jeffrey',PASSWORD('biscuit'));
```

Vous devez utiliser la fonction `PASSWORD()` lorsque vous utilisez la clause `SET PASSWORD`:

```
mysql62; SET PASSWORD FOR jeffrey@"%" = PASSWORD('biscuit');
```

Si vous ajoutez un mot de passe avec la commande `GRANT ... IDENTIFIED BY` ou avec la commande `mysqladmin password`, la fonction `PASSWORD()` n'est pas nécessaire. Ces deux méthodes crypte le mot de passe pour vous :

```
mysql62; GRANT USAGE ON *.* TO jeffrey@"%" IDENTIFIED BY 'biscuit';
```

or

```
shell62; mysqladmin −u jeffrey password biscuit
```

Note: `PASSWORD()` dispose d'une encryption qui n'est pas la même que sous Unix. Il ne faut pas supposer que le mot de passe Unix et ***MySQL*** sont les mêmes, même si `PASSWORD()` retourne la même valeur que celle qui est stockée dans le fichier de mot de passe de Unix. 6.2 Noms et mot de passe des d'utilisateurs MySQL.

6.13 Causes des erreurs "`Access denied`"

L'erreur " `Access denied` " (accès refusé) peut survenir lors d'une tentative de connexion à MySQL. Voici une liste de problème type qui peuvent être la cause d'un tel comportement, et leur solution :

- Avez vous exécuté le script `mysql_install_db` après avoir installé ***MySQL***, pour créer les premières lignes dans les tables de droits ? Si non, faites le. 6.10 Droits initiaux. Testez les droits initaux en exécutant la commande suivante :

```
shell62; mysql −u root test
```

Le serveur doit vous autoriser la connexion sans erreur. Assurez vous aussi que vous avez un fichier appelé ``user.ISD'' dans le dossier de ***MySQL***. Généralement, il se trouve là : ``PATH/var/mysql/user.ISD'', Avec `PATH` qui représente le chemin jusqu'à la racine de l'installation de ***MySQL***.

- Après une installation, il vaut mieux vous connecter au serveur, et mettre en place des utilisateurs et leur permissions.

```
shell62; mysql −u root mysql
```

Le serveur autorisera la connexion car l'utilisateur ***MySQL*** `root` n'a pas de mot de passe à l'origine. Etant donné que cela crée un gros risque au niveau de la sécurité de la base, affecter un mot de passe à l'utilisateur `root` doit être une priorité lorsque vous créez les utilisateurs de votre serveur. Si, lors de la connexion en tant que `root` , vous obtenez l'erreur suivante :

```
Access denied for user: '@unknown' to database mysql
```

Cela signifie que vous n'avez pas de ligne dans la table `user` avec la valeur `'root'` dans la colonne `User` et que `mysqld` n'a pas pu trouver votre hôte dans cette table. Dans ce cas, vous devez redémarrer le serveur avec l'option `−−skip−grant−tables` et éditer les fichiers ``/etc/hosts'' ou ``\windows\hosts'' pour ajouter une ligne pour votre hôte.

- Si vous avez fait la mise à jour depuis une version antérieure à 3.22.11 vers une version 3.22.11 ou plus récente, avez vous exécuté le script `mysql_fix_privilege_tables`? Si non, faites le. La structure des tables de droits a évolé avec la version 3.22.11 lorsque la commande `GRANT` a été ajoutée.
- Si vous avez modifié directement la table de droit (avec les commandes `INSERT` ou `UPDATE`) et que vos modifications n'ont pas été prise en compte, n'oubliez pas qu'il faut exécuter une commande `FLUSH PRIVILEGES` ou `mysqladmin flush−privileges` pour forcer le serveur à recharger les nouvelles valeurs. Sinon, vos modifications n'auront pas d'effet jusqu'au prochain redémarrage du serveur. N'oubliez pas qu'après avoir modifié votre mot de passe `root`, vous n'avez pas à le re entrer jusqu'à ce que vous exécutiez la commande `FLUSH PRIVILEGES` car le serveur n'a pas encore noté que vous l'aviez modifié !
- Si les droits semblent changer durant une session, c'est peut être qu'un super utilisateur les a changé. Recharger les droits affecte non seulement les nouvelles connexions, mais aussi les connexions en cours, comme précisé dans section 6.9 Prise en compte des modifications de droits.
- A fin de tests, démarrez le serveur `mysqld` avec l'option `−−skip−grant−tables`. Ensuite, vous pouvez changer les tables de droits ***MySQL*** et utiliser le script `mysqlaccess` pour vérifier si les modifications ont bien l'effet désiré. Lorsque vous avez fini, lancez la commande `flush−privileges` pour forcer l'utilisation des nouvelles tables de droits. ***Note:*** Recharger les tables annule l'option `−−skip−grant−tables`. Cela vous permet de dire au serveur quand commencer à utiliser les droits, sans le redémarrer.
- Si vous avez des problèmes d'accès avec un programme Perl, Python ou ODBC, essayez de vous connecter au serveur avec `mysql −u user_name nom_base_de_donnees` ou `mysql −u user_name −pyour_pass nom_base_de_donnees`. Si vous etes capable de vous connecter avec le client `mysql`, c'est qu'il y a un problème avec votre programme, et non pas dans les droits d'accès.. (Notez aussi qu'il n'y a pas d'espace entre l'option `−p` et le mot de passe. Alternativement, vous pouvez utiliser `−−password=your_pass` pour spécifier votre mot de passe.)
- Si vous êtes bloqué par votre mot de passe, pensez à utiliser la fonction `PASSWORD()` pour changer de mot de passe avec les commandes `INSERT`, `UPDATE` ou `SET PASSWORD`. Cette fonction n'est pas nécessaire avec la commande `GRANT ... INDENTIFIED BY` ou avec la syntaxe `mysqladmin password`. Reportez vous à la section `Passwords`
- `localhost` est un synonyme qui désigne votre machine locale, ou l'hôte de connexion par défaut. Cependant, les connexions à `localhost` ne sont pas valables si vous utilisez MIT−pthreads (Les connexions `localhost` utilisent les sockets Unix, qui ne sont pas supportées par MIT−pthreads). Pour contourner le problème sur ces systèmes, il préférable d'utiliser l'option `−−host` pour spécifier explicitement le nom de l'hôte. Cela va forcer le type de la connexion à TCP/IP. Dans ce cas, vous devez utiliser le vrai nom d'hôte de votre machine, et l'inscrire dans la table `user` (Cela reste vrai même si vous exécutez un programme client sur la même machine que le serveur).
- Si vous avez l'erreur `Access denied` lors d'une connexion au serveur avec l'option `−u user_name nom_base_de_donnees`, vous pouvez avoir un problème avec la talbe `user`. Vérifiez ceci en exécutant la commande `mysql −u root mysql` puis la commande sui vante :

```
mysql62; SELECT * FROM user;
```

Le résultat devrait contenir une ligne dont les colonnes `Host` et `User` correspondent au nom d'hôte de votre ordinateur et au nom d'utilisateur ***MySQL***.

- Le message `Access denied` apparaîtra si vous essayez de vous connecter, le message d'erreur affichera l'adresse de l'hôte de connexion, le nom d'utilisateur et si vous utilisez un mot de passe. Normalement, il ne devrait y avoir qu'une ligne de la `user` qui corresponde exactement au nom d'hôte de votre ordinateur et au nom d'utilisateur
- Si vous obtenez l'erreur ci dessous lors de la connexion au serveur depuis une autre machine que celle qui heberge le serveur ***MySQL***, c'est qu'il manque une ligne dans la table `user` pour décrire votre machine hôte :

```
Host ... is not allowed to connect to this MySQL server
```

Vous pouvez régler ce problème en utilisant l'utilitaire `mysql` (sur la machine server ! !) pour ajouter une ligne dans la table `user` pour le nom d'hôte de votre ordinateur et le nom d'utilisateur. Si vous ne fonctionnez pas sous ***MySQL*** 3.22 et que vous ne savez pas quelle adresse IP ou quel nom d'hôte est celui de votre machine, il vaut mieux ajouter une ligne avec `'%'` dans la colonne `Host` et redémarrer le serveur avec

l'option `--log` option. Après avoir tenté de vous connecter depuis la machine client, l'historique vous indiquera qui s'est réellement connecté. Vous pouvez alors remplacer la ligne avec `'%'` de la table `user` par une ligne plus spécifique. Sinon, cela peut créer un trou de sécurité.

- Si le test avec `mysql -u root test` fonctionne mais que `mysql -h your_hostname -u root test` retourne une erreur `Access denied`, alors c'est que vous n'avez pas entrer le nom de votre hôte dans la table `user`. Un problème récurent est que la colonne `Host` de la table user contient un nom d'hôte général, mais la résolution de nom retourne un nom de domaine complet, et vice−versa. Par exemple, si vous avec une ligne qui contient `'tcx'` dans la table `user`, mais que votre DNS dit à ***MySQL*** que votre nom d'hôte est `'tcx.subnet.se'`, la ligne ne correspondra pas. Essayez d'ajouter une ligne dans la table `user` qui contienne l'adresse IP numérique de votre hôte, dans la colonne `Host`. (Alternativement vous pouvez aussi ajouter une ligne dans la table `user` avec une valeur dans la colonne `Host` qui contienne un caractère joker (par exemple, `'tcx.%'`). Cependant, c'est une pratique non sécurisée, qui peut provoquer des problèmes de sécurité.
- Si `mysql -u user_name test` fonctionne mais `mysql -u user_name other_nom_base_de_donnees` ne fonctionne pas, c'est que vous n'avez pas d'entrée pour la colonne `other_nom_base_de_donnees` de la table `db`.
- Si `mysql -u user_name nom_base_de_donnees` fonctionne lorsqui'il est exécuté sur la machine server, mais que `mysql -u host_name -u user_name nom_base_de_donnees` ne fonctionne pas lorsqu'il est utilisé sur une machine client, c'est que la machine client n'est pas listée dans la table `user` ou la table `db`.
- Si vous n'arrivez pas à vous defaire de l'erreur `Access denied`, effacez toutes les lignes de la table `user` qui possède un joker (``'%''` ou ``'_'`)'dans la colonne `Host`. Une erreur récurrente est d'insérer une ligne du type `Host='%'` et `User='quidam'`, en pensant que cela va vous permettre de vous connecter depuis la machine `localhost`. La raison qui fait que ceci ne fonctionne pas est que les droits par défaut contiennent une ligne qui est `Host='localhost'` et `User=''`. Or, la ligne qui a `Host ='localhost'` est plus spécifique que `'%'`, et donc, sera utilisée de préférence à celle utilisée lors de la connexion de depuis `localhost`! La procedure correcte est d'insérer un deuxième ligne avec `Host='localhost'` et `User='some_user'`, ou d'effacer les lignes avec `Host='localhost'` et `User=''`.
- Si vous avez l'erreur suivante, c'est que vous avez un problème avec la table `db` ou `host`:

```
Access to database denied
```

Si l'entrée sélectionnée dans la table `db` a une colonne `Host` vide, assurezvous qu'il y a au moins une entrée correspondante dans la table `host` qyu spécifie à quel hôte s'applique ces droits. Si vous avez une erreur en utilisant la commande `SELECT ... INTO OUTFILE` or `LOAD DATA INFILE`, c'est que vous n'avez peut être pas les droits ***file***.

- Rappelez vous que les programmes clients utilisent des paramètres par défaut qui sont stockés dans des fichiers de configuration ou des variables d'environnement. Si un client semble ne pas envoyer les bons paramètres lorsque vous ne les spécifiez pas, allez vérifier les informations stockées dans le fichier ``.my.cnf'` de votre dossier racine, ou les variables d'environnment. Vous pouvez aussi vérifier les paramètres dans les fichiers de configuration de ***MySQL***, mais il est très improbable que des paramètres de connexion soient enregistrés là. 4.15.4 Fichier d'options.
- Si tout ce qui précède a échoué, lancez le démon `mysqld` avec les options de debugging (par exemple, `--debug=d,general,query`). Cette option affichera les hôtes et nom d'utilisateur des connexions, ainsi que des informations concernant les commandes exécutées. Reportez vous à la section G.1 Debugguer un serveur MySQL.
- Si vous avez vraiment tout tenté, et que vous pensez qu'il vous faut le rapporter à la liste de diffusion (en anglais), n'oubliez jamais de fournir un dump des tables de droits de ***MySQL***. Vous pouvez créer ce dump avec l'utilitaire `mysqldump mysql`. Comme toujours, pensez à poster votre problème avec l'utilitaire `mysqlbug`. Dans certains cas, vous devrez redémarrer `mysqld` avec l'option `--skip-grant-tables` pour pouvoir faire le dump.

6.14 Comment protéger MySQL contre les hackers

Lorsque vous vous connectez à un serveur ***MySQL***, vous utiliserez en règle générale un mot de passe. Le mot de passe sera transmis crypté lors de la connexion, et non pas en clair.

Toutes les informations suivantes seront transmises en clair, ce qui fait que quiconque observe la connexion peut lire les informations. Si vous êtes concerné par les problèmes de sécurité, vous pouvez utiliser le protocole compressé (à partir de ***MySQL*** 3.22) pour rendre les choses plus difficile. Pour complexifier encore la lecture, vous pouvez installer `ssh` (http://www.cs.hut.fi/ssh). Ce protocol fourni un cryptage de la connexion TCP/IP entre le client ***MySQL*** et me serveur ***MySQL***.

Pour sécuriser un système ***MySQL***, il est fortement recommandé de suivre les recommandations suivantes :

- Utilisez des mots de passe pour TOUT les utilisateurs ***MySQL***. N'oubliez pas qu'il est très facile de se connecter à la place d'un autre avec la commande `mysql -u other_user nom_base_de_donnees` si `other_user` n'a pas de mot de passe. C'est une stratégie

standard dans les applications clients/serveur. Vous pouvez changer les mots de passe de tous les utilisateurs en éditant le script `mysql_install_db` puis en l'exécutant, ou bien avec la commande `root` suivante :

```
shell62; mysql −u root mysql
mysql62; UPDATE user SET Password=PASSWORD('new_password')
           WHERE user='root';
mysql62; FLUSH PRIVILEGES;
```

- N'utilisez pas ***MySQL*** comme un démon du `root` Unix. `mysqld` peut être exécuté par n'importe quel utilisateur. Vous pouvez aussi créer un nouvel utilisateur `mysql` augmenter la sécurité. Si vous utilisez `mysqld` à partir d'un compte Unix autre que le root, vous n'avez pas à changer l'utilisateur the `root` dans la table `user`, car les noms d'utilisateur ***MySQL*** n'ont rien à voir avec ceux d'Unix. Vous pouvez éditer le script `mysql.server` pour lancer `mysqld` avec un autre utilisateur. Généralement, on le fait avec la commande `su`. Pour plus de détails, @pref{Changing MySQL user}.
- Si vous avez mis le mot de passe du `root` Unix dans le script `mysql.server`, assurez vous que ce fichier n'est lisible que par le `root`.
- Vérifiez que l'utilisateur Unix qui exécute `mysqld` est le seul à avoir les droits d'écriture / lecture dans le dossier des bases de données.
- Ne donnez le droit de ***process*** que très parcimonieusement. L'affichage de la fonction `mysqladmin processlist` dévoile le texte de toutes les commandes en cours d'exécution, ce qui fait que tout ceux qui ont ce droit, ont vue sur les commandes du type `UPDATE user SET password=PASSWORD('not_secure')`. `mysqld` garde toujours une connexion de libre pour les utilisateurs qui ont le droit de ***process***, afin que le `root` ***MySQL*** puisse toujours se connecter et s'assurer du bon fonctionnement des connexions en cours.
- Ne donnez le droit de ***file*** que très parcimonieusement donnez. Quiconque a un tel droit peut écrire un fichier n'importe ou dans le système de fichier, et notamment dans les tables de droits de `mysqld`! Pour sécuriser un peu ceci, tous les fichiers généré par `SELECT ... INTO OUTFILE` ne sont lisibles par tous, mais non modifiables. Le droit de ***file*** peut aussi être utilisé pour lire un fichier quelconque sur le serveur. Cela permet notamment de lire le fichier ``/etc/passwd'' dans une table, et de le lire avec `SELECT`.
- Si vous n'avez pas confiance dans votre serveur DNS, il vaut mieux utiliser les adresse IP numériques, à la place des noms d'hôtes. En principe, l'option `−−secure` de `mysqld` devrait être suffisante pour sécuriser les noms d'hôtes. Dans toues les cas, méfiez vous des jokers dans les noms d'hôtes.

Les options suivantes affectent la sécurité :

- −−secure Les adresse IP retournées par l'appel système `gethostbyname()` sont vérifiées pour s'assurer qu'elle correspondent bien à l'hôte original. Cela rend plus difficile de simuler un hôte. Cette option effectue aussi un nettoyage hygiénique des hôtes. Cette option a été enlevée par défaut dans la version ***MySQL*** 3.21 car elle peut ralentir le serveur, à cause du temps de résolution. ***MySQL*** 3.22 conserve les noms d'hôtes qu'il a déjà résolu dans un cache.
- −−skip−grant−tables Cette option indique au serveur qu'il ne doit pas utiliser les droits enregistrés. Cela donne à tous un accès complet à toutes les bases. (Vous pouvez aussi charger les tables d'un serveur en fonctionnement avec la commande `mysqladmin reload`.)
- −−skip−name−resolve Les noms d'hôtes ne sont pas résolus. Toutes les lignes de la table `Host` doivent contenir des valeurs nuémriques, ou bien `localhost`.
- −−skip−networking N'accepte pas les connexions TCP/. Toutes les connexions à `mysqld` doivent être faites avec les sockets Unix. Cette option n'est pas disponibles pour les systèmes qui utilisent les MIT−pthreads, car MIT−pthreads ne supporte pas les sockets Unix.

7 Programmation MySQL

7.1 Syntaxe des chaînes et nombres

7.1.1 Chaînes

Une chaîne est une séquence de caractères, entourée par des guillemets simples (`'''`) ou doubles(`"""`). Par exemple :

```
'une chaîne'
"une autre chaîne"
```

A l'intérieur d'une chaîne, on trouve des séquences spéciales. Celles−ci commencent avec le caractère backslash (``\''), dit aussi caractère d'échappement. ***MySQL reconnaît les séquences suivantes :***

Within a string, certain sequences have special meaning. Each of these sequences begins with a backslash known as the *escape character*. ***MySQL*** recognizes the following escape sequences:

- `\0` ASCII 0 (`NUL`) le caractère nul.
- `\n` Une nouvelle ligne.
- `\t` Une tabulation.
- `\r` Un retour chariot.
- `\b` Un effacement.
- `\'` Un guillemet simple (``''').
- `\"` Un guillemet double (``"'').
- `\\` Un backslash (``\'').
- `\%` Un pourcentage ``%''. Cela permet de rechercher le caractère `%' dans un contexte ou il pourrait etre considéré comme un caractère spécial.
- `\_` Un souligné ``_'' Cela permet de rechercher le caractère ``_'dans un contexte ou il pourrait etre considéré comme un caractère spécial.'

Il y a plusieurs façons d'introduire des guillemets dans une chaîne.

- Un guillemet simple ``''' dans une chaîne à guillemet simple peut s'écrire : ``''''.
- Un guillemet double ``"'' dans une chaîne à double simple peut s'écrire : ``""''.
- On peut ajouter le caractère d'échappement avant un guillemet : ``\''.
- Un guillemet simple ``''' dans une chaîne à guillemet double ne requiert aucun traitement spécial.
- Un guillemet double ``"'' dans une chaîne à guillemet simple ne requiert aucun traitement spécial.

La commande `SELECT` ci−dessous montre comment fonctionne les guillemets et le caractère d'échappement.

mysql> SELECT 'bonjour', '"bonjour"', '""bonjour""', 'bonj"our', '\'bonjour';

```
+---------+-----------+-------------+----------+----------+
| bonjour | "bonjour" | ""bonjour"" | bonj'our | 'bonjour |
+---------+-----------+-------------+----------+----------+
mysql62; SELECT "bonjour", "'bonjour'", "''bonjour''", "bonj""our", "\"bonjour";
+---------+-----------+-------------+----------+----------+
| bonjour | 'bonjour' | ''bonjour'' | bonj"our | "bonjour |
+---------+-----------+-------------+----------+----------+

mysql62; SELECT "Voici\nQuatre\nLignes\nDistinctes";
+--------------------+
| Voici
Quatre
Lignes
```

```
Distinctes |
+--------------------+
```

Pour ajouter des valeurs binaires dans un BLOB, les caractères suivants doivent être représenté par des séquences spéciales :

- NUL ASCII 0. Représentation : ``\0'' (un backslash et un caractère ASCII ``0'').
- \ ASCII 92, backslash. Représentation : ``\\''.
- ' ASCII 39, guillemet simple. Représentation : ``\''' .
- " ASCII 34, guillemet double. Représentation : ``\"''.
- De préférence, on utilisera une séquence d'échappement pour toute chaîne qui contient un des caractères spéciaux ci−dessus.

7.1.2 Nombres

- Les entiers sont représentés comme une séquence de chiffres. Les nombres réel utilise le point (``.') comme séparateur décimal. Les entiers et les réels peuvent être précédés par le signe moins (`−'), pour indiquer un nombre négatif.'
- Exemples d'entiers :

```
1221
0
−32
```

- Exemples de nombres réels :

```
294.42
−32032.6809e+10
148.00
```

- Lorsqu'un nombre entier est utilisé avec un nombre réel, il est considéré lui aussi, comme un nombre réel.

7.1.3 Valeurs héxadecimales

MySQL supporte les valeurs hexadécimales. Dans un contexte numérique, elles se comportent comme des entiers (précision 64bits). Dans un contexte de chaînes, elles se comportent comme des chaînes binaires dont chaque paire de digits seront converti en caractère.

```
mysql62; SELECT 0xa+0
       −62; 10
mysql62; select 0x5061756c;
       −62; Paul
```

Les chaînes hexadécimales sont souvent utilisées avec ODBC pour donner des valeurs aux colonnes BLOB.

7.1.4 La valeur NULL

- La valeur NULL signifie : ``aucune information". Cette valeur est différente de 0 ou de la chaîne vide.
- NULL est parfois représenté par \N quand on utilise un fichier d'import ou d'export (LOAD DATA INFILE, SELECT ... INTO OUTFILE). Cf section LOAD DATA.

7.1.5 Noms de base de données, table, index, column et alias

- Les noms portés par les base de données, les tables, les index, les colonnes et les alias suivent tous les mêmes règle dans ***MySQL***:
  - Un nom est constitué de caractères alphanumériques et des caractères ``_'' et ``$''.
  - Le nom d'une base de données, d'une table, d'un index ou d'une colonne peut avoir jusqu'à 64 caractères.
  - Un nom peut commencer avec n'importe quel caractère autorisé. En particulier, un nom peut commencer avec un nombre (ce qui n'est pas toujours le cas dans de nombreuses bases de données). Cependant, un nom ne peut pas contenir uniquement des nombres.

- Il est préférable de ne pas utiliser de noms tels que `1e`, car un expression telle que `1e est ambiguë. Le nom peut être interprété comme une expression (1e + 1) ou comme le nombre 1e+1.`
- Il est interdit d'utiliser le point dans les noms, car il est déjà utilisé pour spécifier les noms des colonnes (cf ci dessous).

Avec ***MySQL, il est possible d'accéder aux colonnes d'une table avec les expressions suivantes :*** Il n'est pas nécessaire de préciser `Nom_tab ou Nom_base. Nom_tab pour faire référence à un nom de colonne, à moins que cela puisse être ambiguë. Par exemple, si les tables t1 et t2 contiennent chacune la colonne c, et que lors d'un SELECT, les deux tables soient utilisées. Dans ce cas, c est ambiguë, car elle n'est pas unique dans les tables utilisées, et il faut alors préciser la colonne en précisant t1.c et t2.c. De la même manière, si deux bases de données contiennent chacune une table nommée t, il faudra préciser la base de données utilisée, en notant : db1.t.col1 et db2.t.col2.` Dans la syntaxe `.Nom_tab. Nom_col, on suppose que la table Nom_tab est disponible dans la base de données courante. Cette syntaxe est autorisée pour assurer la compatibilité avec ODBC, car certains programmes compatible ODBC ajoute le préfixe point ``.'' aux noms des tables.`

7.1.5.1 Sensibilité des noms à la casse

- Dans ***MySQL***, les base de données et les tables correspondent à des dossiers et fichiers. Par conséquent, la sensibilité à la casse du système d'exploitation sous−jacent détermine celle de ***MySQL***. Par exemple, les noms de base de données et de tables seront sensible à la casse sous Unix, et pas sous Windows.
- ***N.B.***: Bien que les noms de base de données et de table soient insensible à la casse, sous Windows 32bits, il est préférable de toujours utiliser la même casse pour se référer à un objet dans une même requête. La requête suivante ne fonctionnera pas, car on utilise la même table avec deux noms différents : ma_table et MA_TABLE.

```
SELECT * FROM ma_table WHERE MA_TABLE.col=1;
```

- Dans tous les cas, les noms de colonnes sont insensibles à la casse.
- Les noms d'alias sont sensibles à la casse. La requête suivante ne fonctionnera pas, car elle utilise le même alias, sous la forme : a et A.

```
mysql62; SELECT Nom_col FROM Nom_table AS a
          WHERE a.Nom_col = 1 OR A. Nom_col = 2;
```

- Les alias sur une colonnes sont insensibles à la casse.

7.2 Types de colonnes

- ***MySQL*** dispose d'un grand nombre de type de colonnes. Ces types peuvent être regroupés en trois catégories : les types numériques, les types date et heure, et les types chaînes de caractères. Ce paragraphe présente les différents types disponibles et leur tailles respectives, puis il détaille les caractéristiques de chaque catégorie. La présentation des types est intentionnellement brèves : les descriptions détaillées fourniront toutes les informations nécessaires pour un type particulier, comme par exemple les formats autorisés pour chaque colonne.
- Voici la liste des types de colonnes utilisés par ***MySQL.*** Les codes suivants sont utilisé durant les descriptions :
- `M` Indique la taille maximale d'affichage. La taille maximale autorisée par défaut est 255.
- `D` S'applique aux types à virgule flottante, et précise le nombre de chiffre après la virgule.

Crochets (``[`` et ``]``) indique que cet argument est optionnel.

NB : Il est toujours possible de spécifier `ZEROFILL` pour une colonne. ***MySQL*** ajoutera alors automatiquement l'attribut `UNSIGNED` à la colonne.

- `TINYINT[(M)] [UNSIGNED] [ZEROFILL]` Un très petit entier. Signé, il couvre l'intervalle `−128` à `127` ; non signé, il couvre `0` à `255`.
- `SMALLINT[(M)] [UNSIGNED] [ZEROFILL]` Un petit entier. Signé, il couvre l'intervalle `−32768` à `32767`; non signé, il couvre `0` à `65535`.
- `MEDIUMINT[(M)] [UNSIGNED] [ZEROFILL]` Un entier de taille intermédiaire. Signé, il couvre l'intervalle `−8388608` à `8388607`; non signé, il couvre `0` à `16777215`.
- `INT[(M)] [UNSIGNED] [ZEROFILL]` Un entier de taille normale. Signé, il couvre l'intervalle `−2147483648` à `2147483647`; non signé, il couvre `0` à `4294967295`.
- `INTEGER[(M)] [UNSIGNED] [ZEROFILL]` Un synonyme pour `INT`.
- `BIGINT[(M)] [UNSIGNED] [ZEROFILL]` Un entier de grande taille. Signé, il couvre l'intervalle `−9223372036854775808` à `9223372036854775807`; non signé, il couvre `0` à `18446744073709551615`. NB : toutes les opérations arithmétiques effectuée en interne, utilise des `BIGINT` signés ou `DOUBLE` , donc il ne faut pas utiliser les grands entiers non signé au delà de `9223372036854775807` (63 bits), hormis pour les fonctions sur les bits. NB : les opérations `−`, `+` et * utiliseront des BIGINT , même lorsque les arguments seront des entiers. Cela signifie que la multiplication de deux grands entiers (ou le résultat de fonction qui retourne des entiers) peut avoir des résultats surprenants, si le résultat est supérieur à 9223372036854775807
- `FLOAT(precision) [ZEROFILL]` Un nombre à virgule flottante. Il est obligatoirement signé. `precision` peut prendre les valeurs de 4 ou 8.

FLOAT(8) est un nombre à précision double. Ces types sont identiques aux types FLOAT et DOUBLE décrit ci−dessous, mais leur précision peut être paramétrée. Avec ***MySQL*** 3.23, ce sont de vrais nombres à virgule flottante, alors qu'avec les anciennes versions, FLOAT(precision) n'avait que 2 décimales. Cette syntaxe a été ajoutée pour assurer la compatibilité ODBC.

- FLOAT[(M,D)] [ZEROFILL] Un nombre à virgule flottante, en précision simple. Il est toujours signé. Les valeurs sont comprises −3.402823466E+38 et −1.175494351E−38.
- DOUBLE[(M,D)] [ZEROFILL] Un nombre à virgule flottante, en précision double. Il est toujours signé. Les valeurs sont comprises −1.7976931348623157E+308 et 2.2250738585072014E−308.
- DOUBLE PRECISION[(M,D)] [ZEROFILL]
- REAL[(M,D)] [ZEROFILL] Des synonymes pour DOUBLE.
- DECIMAL(M,D) [ZEROFILL] Un nombre à virgule flottante. Il est toujours signé. Il se comporte comme une colonne CHAR . Il n'est pas paqué, c'est à dire que le nombre est stocké comme une chaîne de chiffre. Chaque chiffre, le signe moins, la virgule occupe un caractère. Si D vaut 0, le nombre n'aura pas de décimales, ni de virgule. La taille maximale pour les décimales est la même que pour les DOUBLE , mais en il peut être limité par le choix de of M et D. Avec ***MySQL*** 3.23, M n'inclut plus le signe moins ``−''. ni la virgule des nombres décimaux (norme ANSI SQL.).
- NUMERIC(M,D) [ZEROFILL] Un synonyme pour DECIMAL.
- DATE Une date. L'intervalle valide de date va de '1000−01−01' à '9999−12−31'. ***MySQL*** affiche les DATE avec le format , mais il est possible d'affecter des DATE en utilisant indifféremment des chaînes ou des nombres.
- DATETIME Une combinaison de date et d'heure. L'intervalle valide va de '1000−01−01 00:00:00' to '9999−12−31 23:59:59'. ***MySQL*** affiche DATETIME avec le format 'YYYY−MM−DD HH:MM:SS', mais il est possible d'affecter des DATETIME ***en utilisant indifféremment des chaînes ou des nombres.***
- TIMESTAMP[(M)] Un timestamp : la date et l'heure, exprimée en secondes, depuis le 1er janviers 1970. Il permet de couvrir un intervalle allant de'1970−01−01 00:00:00' à quelque part, durant l'année 2037. ***MySQL*** affiche les TIMESTAMP avec les format YYYYMMDDHHMMSS, YYMMDDHHMMSS, YYYYMMDD ou YYMMDD, suivant que M vaut 14 (ou absent), 12, 8 ou 6, mais il est possible d'affecter des TIMESTAMP en utilisant indifféremment des chaînes ou des nombres. Une colonne de type TIMESTAMP est très pratique pour enregistrer des dates et heures lors d'un INSERT ou UPDATE, car cette colonne sera automatiquement mis à la date et heure de l'opération la plus récente, si aucune valeur n'est précisée. Il est aussi possible d'affecter l'heure courante en assignant la valeur NULL à une colonne de type . (7.2.6 Types date et heure)
- TIME Une mesure de l'heure. L'intervalle valide est '−838:59:59' à '838:59:59'. ***MySQL*** affiche TIME au format 'HH:MM:SS', mais il est possible d'affecter des TIME en utilisant indifféremment des chaînes ou des nombres.
- YEAR Un an. L'intervalle valide est 1901 à 2155, et 0000. ***MySQL*** affiche YEAR au format YYYY, mais il est possible d'affecter des YEAR en utilisant indifféremment des chaînes ou des nombres}(Le type YEAR est nouveau en ***MySQL*** 3.22.)
- CHAR(M) [BINARY] Une chaîne de caractère de taille fixe, et toujours complétée à droite par des espaces. M va de 1 à 255 caractères. Les espaces supplémentaires sont supprimés lorsque la valeur est retournée dans une requête. Les tris et comparaisons effectués sur des valeurs de type CHAR sont insensibles à la casse, à moins que le mot clé BINARY soit précisé.
- VARCHAR(M) [BINARY] Une chaîne de caractère de longueur variable. Les espaces en fin de chaîne sont supprimés lorsque la chaîne est stockée (ce n'est pas conforme à la norme ANSI SQL). Va de 1 a 255 caractères . Les tris et comparaisons effectués sur des valeurs de type VARCHAR sont insensibles à la casse, à moins que le mot clé BINARY soit précisé. Voir aussi la section 7.6.1 Modifications automatiques de type de colonne.
- TINYBLOB
- TINYTEXT Un objet BLOB ou TEXT avec une longueur maximale de 255 (2^8 − 1). Voir aussi la section 7.6.1 Modifications automatiques de type de colonne.
- BLOB
- TEXT Un objet BLOB ou TEXT avec une longueur maximale de 65535 (2^16 − 1). 7.6.1 Modifications automatiques de type de colonne.
- MEDIUMBLOB
- MEDIUMTEXT Un objet BLOB ou TEXT avec une longueur maximale de 16777215 (2^24 − 1). 7.6.1 Modifications automatiques de type de colonne.
- LONGBLOB
- LONGTEXT Un objet BLOB ou TEXT avec une longueur maximale de 4294967295 (2^32 − 1). 7.6.1 Modifications automatiques de type de colonne.
- ENUM('value1','value2',...) Une énumération. Un objet chaîne peut prendre une des valeurs contenue dans une liste de valeur 'value1', 'value2', ..., ou NULL . Une ENUM peut avoir un maximum de 65535 valeurs distinctes.
- SET('value1','value2',...) Un ensemble. Un objet chaîne peut prendre une ou plusieurs valeurs, chacun de ces valeur devant être contenue dans une liste de valeurs 'value1', 'value2', ... Un SET peut prendre jusqu'à 64 éléments.

7.2.1 Tailles nécessaires pour le stockage de types de colonnes

Voici la liste des espaces mémoire requis, par type.

7.2.2 Types numériques

Column type	*Storage required*
TINYINT	1 byte
SMALLINT	2 bytes
MEDIUMINT	3 bytes
INT	4 bytes
INTEGER	4 bytes

BIGINT	8 bytes
FLOAT(4)	4 bytes
FLOAT(8)	8 bytes
FLOAT	4 bytes
DOUBLE	8 bytes
DOUBLE PRECISION	8 bytes
REAL	8 bytes
DECIMAL(M,D)	M bytes (D+2, if M < D)
NUMERIC(M,D)	M bytes (D+2, if M < D)

7.2.3 Types date et heure

Column type	*Storage required*
DATETIME	8 bytes
DATE	3 bytes
TIMESTAMP	4 bytes
TIME	3 bytes
YEAR	1 byte

7.2.4 Types chaîne

Column type	*taille requise*
CHAR(M)	M octets, 1 <= M <= 255
VARCHAR(M)	L+1 bytes, avec L <= M et 1 <= M <= 255
TINYBLOB, TINYTEXT	L+1 octets, where L < 2^8
BLOB, TEXT	L+2 octets, where L < 2^16
MEDIUMBLOB, MEDIUMTEXT	L+3 octets, where L < 2^24
LONGBLOB, LONGTEXT	L+4 octets, where L < 2^32
ENUM('value1','value2',...)	1 ou 2 octets, suivant le nombre de valeur dans l'énumeration (65535 au maximum)
SET('value1','value2',...)	1, 2, 3, 4 ou 8 octets, suivant le nombre de membre de l'ensemble (64 membres au maximum)

Les types VARCHAR, BLOB et TEXT sont des types à longueur variable, dont taille de stockage dépend plus de la valeur qui leur est assignée que de leur taille maximale. Par exemple, une colonne de type VARCHAR(10) peut contenir une chaîne de 10 caractères aux maximum. La taille réelle nécessaire est la longueur de la chaîne, plus 1 octet, qui stockera la taille réelle de la chaîne. Par exemple, la chaîne 'abcd' occupe 5 octets.

Les types BLOB et TEXT ont besoin de 1, 2, 3 ou 4 octets pour stocker la taille de la colonne, en fonction du type.

Si une table possède au moins une colonne de longueur variable, l'enregistrement sera aussi de longueur variable. Il faut noter que lorsqu'une table est créée, ***MySQL*** peut, sous certaines conditions, changer le type d'une colonne de longueur variable en un type de colonne de longueur fixe, et vice−versa. Pour plus de détails, 7.6.1 Modifications automatiques de type de colonne.

La taille d'un objet ENUM est déterminé par le nombre d'énumération différentes. 1 octet est suffisant pour

décrire une énumération qui a jusqu'à 255 valeurs différentes ; 2 octets sont nécessaires décrire une énumération qui aurait jusqu'

La taille d'un objet SET est déterminé par le nombre d'élément distinct qu'il contient. Si la taille d'un SET est N, le SET occupera (N+7)/8 octets, arrondi aux entiers 1,2,3,4, ou 8 octets. Un ensemble peut contenir jusqu'à 64 éléments.

7.2.5 Types numériques

Tous les types entiers disposent d'un attribut optionnel UNSIGNED. Les valeurs non signées sont utilisées quand les nombres utilisé sont uniquement positifs, ou bien lorsqu'il faut pouvoir manipuler des nombres un peu plus grand de normalement.

Tous les types numériques disposent d'un attribut optionnel ZEROFILL. Cette option force l'affichage de tous les zéros non significatifs. Ainsi, dans une colonne de type INT(5) ZEROFILL, 4 sera affiché : 00004.

Quand une valeur trop grande est affectée à une colonne, ***MySQL*** limitera cette valeur au maximum qu'il peut stocker dans la colonne.

Par exemple, soit une colonne de type INT qui accueille des nombres dans l'intervalle −2147483648 to 2147483647. Lorsqu'on tente d'inserer −9999999999 dans cette colonne, ***MySQL*** utilisera automatiquement la plus petite valeur possible, soit −2147483648 . De même, Lorsqu'on tente d'inserer 9999999999 dans cette colonne, ***MySQL*** utilisera automatiquement la plus grande valeur possible, soit 2147483647.

Si une colonne est de type INT UNSIGNED, la taille de la colonne est la même, mais les extrémités sont différentes. Lors d'une tentative d'insertion, −9999999999 et 9999999999 deviendront respectivement 0 et 4294967296.

Ces conversions implicites sont signalées comme des alertes (``warnings''), lors des requêtes incluant ALTER TABLE, LOAD DATA INFILE, UPDATE et INSERT multi−lignes.

La taille maximale (M) et le nombre de décimales (D) sont utilisées lors du formatage et des calculs de la taille maximale d'une colonne.

MySQL tentera d'enregistrer n'importe quelle valeur, du moment que cette dernière peut être contenue dans la colonne, et malgré le dépassement de limite d'affichage. Par exemple, une colonne de type INT(4) peut afficher 4 caractères. Il est cependant possible d'insérer une valeur qui a plus que 4 chiffres, telle que 12345. La taille d'affichage est dépassée, mais 12345 est bien dans l'intervalle autorisé pour un INT. Donc, ***MySQL*** va enregistrer la valeur de 12345 . Lors d'une requête, ***MySQL*** retournera bien la vraie valeur, c'est à dire . 12345

Le type DECIMAL peut être considéré comme un type numérique (puisqu'il est synonyme de NUMERIC), mais ce type est en fait enregistré comme une chaîne. Un caractère est utilisé pour chaque chiffre, pour la virgule et pour le signe moins ``−''. Si D vaut 0, DECIMAL et NUMERIC ne contiennent ni virgule, ni partie décimale.

La taille maximale d'une valeur DECIMAL est la même que celle d'un DOUBLE, mais elle peut dépendre des choix de M et D. Par exemple, un DECIMAL déclaré tel que indique DECIMAL(4,2) que la valeur maximale aura 2 chiffres après la virgule. Etant donné la façon avec laquelle le type DECIMAL est enregistré, ce

DECIMAL sera compris entre−.99 to 9.99, ce qui est nettement moins que les valeurs accessibles avec un DOUBLE.

Pour éviter certains problèmes d'arrondissement, ***MySQL*** ajuste toujours les valeurs qu'il enregistre au nombre de décimale de la colonne. Ainsi, pour une colonne de type FLOAT(8,2)., le nombre de décimale est 2. Donc, un nombre tel que 2.333 sera arrondi à 2.33, puis enregistré.

7.2.6 Types date et heure

Les types date et heure sont DATETIME, DATE, TIMESTAMP, TIME et YEAR. Chacun dispose d'un intervalle de validité, et une valeur ``zéro'', qui peut être utilisé pour indiquer une valeur illégale.

Voici quelques considérations générales à garder à l'esprit quand on travaille avec les types date et heure :

- ***MySQL*** retourne les valeurs de date et d'heure dans un format standard unique, mais il est capable d'interpréter un grand nombre de format d'entrée. Néanmoins, seuls les formats décrit dans les sections suivantes sont supportés. ***MySQL*** attends des dates valides, et des effets imprévisibles peuvent résulter de l'utilisation d'autres formats.
- Bien que ***MySQL*** tente d'interpréter un grand nombre de format de date, l'année devra toujours être placée à gauche. Les dates doivent être données dans l'ordre année−mois−jour (ie : '98−09−04'), plutôt que dans l'ordre mois−jour−année ou l'ordre jour−mois−année utilisés habituellement. (ie '09−04−98', '04−09−98').
- ***MySQL*** convertit automatiquement une date ou une heure en un nombre, si cette valeur est utilisé dans un contexte numérique, et vice−versa.
- ***Quand MySQL rencontre une valeur pour une date ou une heure qui n'est pas valide, il la convertit en valeur*** ``zéro''. Les problèmes de dépassement de capacités sont réglés comme pour les types numériques, en ramenant la valeur au maximum ou au minimum de l'intervalle autorisé. La table suivante montre les formats des valeurs ``zéro''.

Column type	***``Zero'' value***
DATETIME	'0000−00−00 00:00:00'
DATE	'0000−00−00'
TIMESTAMP	00000000000000 (length depends on display size)
TIME	'00:00:00'
YEAR	0000

- Les valeurs ``zéro'' sont des valeurs particulières, mais il est parfaitement possible de les utiliser pour les enregistrer ou y faire référence. Il est aussi possible de les écrire '0' ou 0,qui sont plus facile à écrire.
- Les dates ``Zéro'' utilisées via ***MyODBC*** sont automatiquement converties à NULL par ***MyODBC*** 2.50.12 et mieux, car ODBC ne peut pas manipuler de telle valeur.

7.2.6.1 Bug de l'an 2000 et données de types date

Les types DATETIME, DATE et TIMESTAMP sont proches. Cette section décrit leur caractéristiques, et montre en quoi ils sont similaires, et en quoi ils sont différents.

Le type DATETIME est utile pour manipuler en même temps une date et une heure. . ***MySQL*** retourne et affiche les valeurs de type DATETIME au format 'YYYY−MM−DD HH:MM:SS'. L'intervalle valide pour le type DATETIME est '1000−01−01 00:00:00' à '9999−12−31 23:59:59'. (``valide`` signifie que des valeurs anciennes pourrait fonctionner, mais qu'il n'y a aucune garantie).

Le type DATE est utilisé pour manipuler simplement une date, sans l'heure. ***MySQL*** retourne et affiche les valeurs de type DATE au format 'YYYY−MM−DD' L'intervalle valide pour le type DATE est '1000−01−01' à '9999−12−31'

Le type TIMESTAMP est utilisé automatiquement lors de requête , avec la valeur courante de date et d'heure. Si il y a plusieurs colonnes de type TIMESTAMP , seule la première sera automatiquement mise à jour.

La datation automatique intervient sur la première colonne de type `TIMESTAMP`, et dans les conditions suivantes :

- La colonne n'est pas expressément nommée dans la requête `INSERT` ou `LOAD DATA`
- La colonne n'est pas expressément nommée dans une requête , et au moins une autre colonne change `UPDATE` de valeur (Il est important de noter qu'une requête `UPDATE` qui affecterait la même valeur que celle qui est déjà affectée dans la colonne, alors la colonne `TIMESTAMP` ne sera pas mise à jour, car ***MySQL*** va ignorer la requête, par souci d'efficacité.)
- La colonne `TIMESTAMP` est explicitement mise à `NULL`.

Dans le cas où il y a plusieurs colonnes de type `TIMESTAMP`, il est possible d'affecter la valeur courante de date et d'heure, en affectant la valeur `NULL` ou `NOW()`.

Il est possible d'affecter n'importe quelle date à une colonne de type `TIMESTAMP`, en lui affectant explicitement la valeur désirée. Cela est vrai pour toutes les colonnes de type `TIMESTAMP`, y compris la première. Par exemple, il est possible d'utiliser une colonne de type pour enregistrer le moment de création de la ligne, mais qui ne sera plus changé par la suite.:

- Il suffit de laisser ***MySQL*** affecter la valeur automatiquement lors de la création de la ligne. L'initalisation sera faite au jour et heure courante.
- Lors des modifications ultérieurs, il suffit d'affecter à la colonne sa propre valeur.

Ou alors, il est aussi simple d'utiliser une colonne de type `DATETIME`, qui sera initialisé à l'aide de la commande `NOW()`, et qui ne sera plus jamais modifiée.

`TIMESTAMP` couvre un espace de temps qui commence en 1970 et se termine quelques par en 2037, avec un précision d'une seconde. Ses valeurs sont affichée comme des nombres.

Le format utilisé par pour retourner les valeurs de type dépendent de la taille de l'affichage, comme présenté ci dessous. Un complet affiche 14 chiffres, mais il est possible de n'afficher qu'une partie.

Column type	***Display format***
`TIMESTAMP(14)`	`YYYYMMDDHHMMSS`
`TIMESTAMP(12)`	`YYMMDDHHMMSS`
`TIMESTAMP(10)`	`YYMMDDHHMM`
`TIMESTAMP(8)`	`YYYYMMDD`
`TIMESTAMP(6)`	`YYMMDD`
`TIMESTAMP(4)`	`YYMM`
`TIMESTAMP(2)`	`YY`

Toutes les colonnes de type ont besoin de la même quantité de mémoire, quelque soit le format d'affichage. Les formats d'affichage les plus courants sont le 6, le 8, le 12 et le 14. Il est possible d'imposer une taille arbitraire au format d'affichage au moment de la création de la table, mais 0 et toutes les valeurs supérieures à 14 seront ramenées à 14. Les valeurs impaires entre 1 et 13 sont arrondies au nombre entier pair supérieur.

Il est possible d'affecter des valeurs de type `DATETIME`, `DATE` et `TIMESTAMP` en utilisant n'importe lequel des formats suivants :

- Une chaîne, avec le format `'YYYY−MM−DD HH:MM:SS'` ou `'YY−MM−DD HH:MM:SS'`. Les '−' et ':' ne sont pas obligatoires, et n'importe quel caractère non numérique. Par exemple, `'98−12−31 11:30:45'`, `'98.12.31 11+30+45'`, `'98/12/31 11*30*45'` et `'98@12@31 11^30^45' sont équivalents.`
- Une chaîne, avec le format `'YYYY−MM−DD'` ou `'YY−MM−DD'`. Les '−' et ':' ne sont pas obligatoires, et ils peuvent être remplacé par quel caractère non numérique. Par exemple, `'98−12−31'`, `'98.12.31'`, `'98/12/31'` et `'98@12@31' sont équivalents.`
- Une chaîne sans aucun délimiteur, avec le format `'YYYYMMDDHHMMSS'` ou `'YYMMDDHHMMSS'`, en supposant que cette chaîne a bien un

sens en tant que date. Par exemple, '19970523091528' et '970523091528 seront comprises comme '1997−05−23 09:15:28'.Au contraire, '971122459015' n'est pas valide (le nombre de seconde n'est pas valide), et sera remplacé par : '0000−00−00 00:00:00'.
- Une chaîne sans aucun délimiteur, avec le format 'YYYYMMDD' or 'YYMMDD', en supposant que cette chaîne a bien un sens en tant que date. Par exemple, '19970523' et '970523' seront comprises comme '1997−05−23'.Au contraire, '971332' n'est pas valide (le nombre de mois et de jour ne sont pas valides), et sera remplacé par : '0000−00−00'.
- Un nombre avec le format YYYYMMDDHHMMSS or YYMMDDHHMMSS, en supposant que cette chaîne a bien un sens en tant que date. Par exemple, 19830905132800 et 830905132800 seront comprises comme '1983−09−05 13:28:00'
- Un nombre avec le format YYYYMMDD or YYMMDD, en supposant que cette chaîne a bien un sens en tant que date. Par exemple, 19830905132800 et 830905132800 seront comprises comme '1983−09−05 13:28:00'
- Le résultat d'une fonction qui retourne une valeur acceptable en dans un contexte de DATETIME, DATE or TIMESTAMP, comme NOW() or CURRENT_DATE.

Toutes les valeurs de type DATETIME, DATE ou TIMESTAMP sont converties automatiquement en ``zero'' du même type ('0000−00−00 00:00:00', '0000−00−00' ou 00000000000000).

Pour les valeurs spécifiées au format chaîne avec des délimiteurs, il n'est pas nécessaire de préciser les deux chiffres pour les mois ou les jours. Par exemple, '1979−6−9' et '1979−06−09' sont identiques. De la même façon, pour les valeurs spécifiées au format chaîne avec des délimiteurs, il n'est pas nécessaire de préciser les deux chiffres pour les heures, minutes ou secondes. Ainsi, '1979−10−30 1:2:3' et '1979−10−30 01:02:03' sont identiques.

Les valeurs spécifiée au format nombre doivent avoir 6, 8, 12 ou 14 chiffres. Si le nombre a 8 ou 14 chiffres, ***MySQL*** utilisera respectivement le format YYYYMMDD ou YYYYMMDDHHMMSS, où l'année sera représentée par les quatre premiers chiffres. Si le nombre a 6 ou 12 chiffres, ***MySQL*** utilisera respectivement le format YYMMDD or YYMMDDHHMMSS, où l'année sera représentée par les deux premiers chiffres. Si le nombre n'a pas le bon nombre de chiffre, il sera complété avec des zéros non significatif (placés avant le premier chiffre), jusqu'à correspondre à une taille analysable.

Les valeurs non délimitées, sont interprétées en fonction de leur longueur. Si la chaîne fait 8 ou 14 caractères, l'année sera considérée comme ayant 4 chiffres. Sinon, l'année n'aura que les deux premiers chiffres. La chaîne est interprétée de gauche à droite, en lisant successivement l'année, le mois, le jour, l'heure, la minute et la seconde, dans la mesure où la chaîne est suffisamment longue. Cela implique qu'il ne faut pas utiliser de chaîne de moins de 6 caractères. Par exemple, la chaîne '9903', qui pourrait s'interpréter mars 1999, sera remplacée par la date ``zéro''. En effet, est capable de trouver l'année (99) et le mois (03) mais pas le jour (00), et donc cette valeur n'est pas valide.

Les colonnes de type TIMESTAMP enregistre les dates avec la précision maximum, quelque soit la taille d'affichage. Cela a plusieurs implications :

- Il faut toujours préciser l'année, le mois et le jour, même si le type de la colonne est TIMESTAMP(4) ou TIMESTAMP(2). Sinon, la valeur ne sera pas valide, et donc, remplacée par ``zéro''.
- Si la structure de la table est modifié avec , pour agrandir la colonne, l'affichage présentera des données qui étaient précédemment cachées.
- De la même façon, réduire la taille d'affichage d'une colonne TIMESTAMP ne causera pas de perte d'information : elles seront simplement cachées.
- Bien que les valeurs TIMESTAMP soient en précision maximale, la seul fonction qui opère directement sur la valeur enregistrée dans la base est UNIX_TIMESTAMP(). Les autres fonctions utilisent la valeur formatée. Cela signifie qu'il n'est pas possible d'utiliser une fonction telle que HOUR() ou SECOND() à moins que la partie significative de TIMESTAMP soit inclus dans la valeur formatée. Par exemple, la partie HH d'une valeur de type TIMESTAMP ne sera pas affichée, à moins que la taille de l'affichage soit au moins de 10. Ainsi, utiliser la fonction sur une valeur de type de taille inférieure à 10 risque de retourner une valeur incohérente.

Il est possible d'affecter des valeurs de type date dans une variable d'un autre type date. Cependant, cela peut engendrer des altérations ou des pertes d'informations. :

- Lors de l'affectation d'une DATE dans une variable de type DATETIME ou TIMESTAMP, la partie heure du résultat est mis à '00:00:00', car la valeur DATE ne contient aucune information d'heure.
- Lors de l'affectation d'une DATETIME ou TIMESTAMP dans une variable de type DATE, la partie heure du résultat est perdue , car la valeur DATE ne contient aucune information d'heure.

- Il ne faut pas confondre les différents formats de spécifications des DATETIME, DATE et TIMESTAMP . En effet, pour le même format d'acquisition, les différents types n'ont pas le même intervalle de validité. Par exemple, TIMESTAMP ne peut contenir de valeur antérieure à 1970 ou postérieure à 2037. Cela signifie qu'une date telle que as '1968−01−01', tout en étant valide pour les types DATETIME or DATE, n'est pas valide pour le type TIMESTAMP et sera transformée en 0, lors de l'affectation à un tel objet.

Attention aussi aux erreurs de spécifications :

- En utilisant la forme de spécification avec délimiteurs, comme les délimiteurs ne sont pas imposé, il est possible d'être induit en erreur par la forme : Ainsi, '10:11:12' ressemble à une heure, à cause des délimiteurs " : ". Mais, utilisé dans un contexte de date, il peut aussi être interprété comme la date'2010−11−12'. De même, la valeur '10:45:15' sera convertie en 0, car 45 ,'est pas un mois valide.
- Les années spécifiées sur deux chiffres sont ambiguës :
  - Les années qui sont dans l'intervalle 00−69 sont considéré comme 2000−2069.
  - Les années qui sont dans l'intervalle 70−99 sont considéré comme 1970−1999.

7.2.6.2 Le type TIME

MySQL retourne et affiche les valeurs de type TIME aux formats 'HH:MM:SS' ou 'HHH:MM:SS'. Les valeurs de type TIME vont de '−838:59:59' à '838:59:59'. Le nombre d'heure peut être rendu aussi grand afin de pouvoir représenter les heures du jour, mais aussi de faire des calculs de différence d'heure entre deux jours, ce qui conduit à des durée très supérieures à 24h, voire même des valeurs négatives.

Les valeurs de type TIME peuvent être définies de nombreuses manières différentes :

- Une chaîne de format 'HH:MM:SS'. Les :' ne sont pas obligatoires, et ils peuvent être remplacé par n'importe quel caractère non numérique. Par exemple, , '10:11:12' et '10.11.12'sont équivalents. .
- Une chaîne sans délimiteurs, au format'HHMMSS', en supposant qu'elle est un sens en tant que date. Par exemple, '101112' sera interpreté comme '10:11:12', mais '109712' n'est pas valide et deviendra '00:00:00'.
- Un nombre au format HHMMSS, en supposant que cela ait un sens. Par exemple, 101112 vaudra . '10:11:12'
- Le résultat d'une fonction qui retourne une valeur acceptable en dans un contexte de DATETIME, DATE or TIMESTAMP, comme NOW( ) or CURRENT_DATE.

Pour les valeurs de type TIME spécifiées au format chaîne avec des délimiteurs, il n'est pas nécessaire de préciser les deux chiffres pour les heures, minutes ou secondes. Ainsi, '8:3:2' et '08:03:02'.sont identiques.

Attention aux affections de valeurs courtes dans une colonne de type TIME. ***MySQL*** interprète les valeurs en supposant que les chiffres de gauche sont les secondes (***MySQL*** interprète les valeurs de type TIME comme des intervalles de temps, plutôt qu'une date). Par exemple, '11:12', '1112' et 1112 pourraient être confondues avec '11:12:00' (12 minutes après 11 heures), mais ***MySQL*** le comprend comme '00:11:12' (11 minutes, 12 seconds). De même, '12' et 12 representent '00:00:12'.

Toutes les valeurs de TIME, qui sont hors de l'intervalle de validité sont ramenées à la valeur valide la plus proche. Ainsi, '−850:00:00' et '850:00:00' sont respectivement converties en'−838:59:59' et '838:59:59'.

Toutes les valeurs invalides de TIME sont converties en '00:00:00'. Il faut bien savoir que '00:00:00'est une valeur de TIME valide. Ainsi, si est stocké dans une table, il est impossible de dire si cela provient d'une erreur, ou si il a été affecté à cette valeur.

7.2.6.3 Le type YEAR

Le type YEAR sert à représenter les années sur un octet.

MySQL retourne et affiche les YEAR au format YYYY: L'intervalle de validité est de 1901 à 2155.

Les valeurs de type YEAR peuvent être définies de nombreuses manières différentes :

- Une chaîne de 4 chiffres, dans l'intervalle de `1901` à `2155`.
- Un nombre de 4 chiffres, dans l'intervalle de `1901` à `2155`.
- Une chaîne de 2 chiffres, dans l'intervalle de '`00`' à '99'. Les valeurs de `'00'` à `'69'` et de`'70'` à `'99'` seront converties en valeurs de type YEAR, dans les intervalles respectifs de `2000` à `2069` et de `1970` à `1999`.
- Un nombre de 2 chiffres, dans l'intervalle de `1` à 99. Les valeurs de `1` à `69` et de`70` à `99` seront converties en valeurs de type YEAR, dans les intervalles respectifs de `2000` à `2069` et de `1970` à `1999`. Il faut bien noter que ce format diffère légèrement du précédent, car il n'est pas possible de passer un nombre égal à 0, pour obtenir l'année 2000. Il faut spécifier une chaîne, `'0'` ou `'00'`, sinon ***MySQL*** retournera `0000`
- Le résultat d'une fonction qui retourne une valeur acceptable en dans un contexte de `DATETIME`, `DATE` or `TIMESTAMP`, comme `NOW()`.

Toutes les valeurs invalides de YEAR sont converties en `0000`.

7.2.7 Types chaîne

Les types chaînes sont CHAR, VARCHAR, `BLOB`, `TEXT`, `ENUM` et `SET`.

7.2.7.1 Les types CHAR et VARCHAR

Les types CHAR et VARCHAR sont similaires, mais ils diffèrent par la manière dont ils sont stockés.

Les valeurs de type CHAR sont de longueur fixée. La longueur est déclarée lors de la création de la table. Cette longueur peut aller de 1 à 255. Quand une valeur de type CHAR est enregistrée, elle est complétée à gauche par des espaces. Lorsque ***MySQL*** retourne cette valeur, ces espaces sont effacés.

Les valeurs de type VARCHAR sont de longueur variable. Une longueur maximum est déclarée lors de la création de la table. Cette longueur peut aller de 1 à 255. Quand une valeur de type CHAR est enregistrée, seul les caractères utiles sont enregistrés, plus un octet pour enregistrer la taille de la chaîne. Les ne sont pas complétée à gauche par des espaces, mais les espaces en début de valeurs sont effacés lors de l'enregistrement (cette fonction d'effacement des espaces à l'enregistrement n'est pas spécifiée dans ANSI SQL).

Lors de l'affectation d'une valeur de type CHAR ou VARCHAR dans une colonne trop petite, la valeur est tronquée à la taille de la colonne.

La table ci dessous illustre les différences de taille entre les deux types :

Value	`CHAR(4)`	***Storage required***	`VARCHAR(4)`	***Storage required***
`''`	`'  '`	4 bytes	`''`	1 byte
`'ab'`	`'ab '`	4 bytes	`'ab'`	3 bytes
`'abcd'`	`'abcd'`	4 bytes	`'abcd'`	5 bytes
`'abcdefgh'`	`'abcd'`	4 bytes	`'abcd'`	5 bytes

Les valeurs retournées seront les mêmes, car dans tous les cas, les espaces situés en début de chaîne seront effacées

Les valeurs de type CHAR et VARCHAR sont triées et comparées sans tenir compte de la casse, à moins que l'option `BINARY` ai été activée lors de la création de la table. Cette option signifie que cette colonne sera triée et comparée en tenant compte de la casse, et en fonction de l'ordre de la table ASCII de la machine qui supporte ***MySQL***.

Le mode est contagieux : Cela signifie que si une colonne de type binaire est utilisée dans une expression, la comparaison tiendra compte de la casse.

MySQL peut changer spontanément les types de colonne lors de la création d'une table.. 7.6.1 Modifications automatiques de type de colonne.

7.2.7.2 Les types BLOB et TEXT

Un BLOB est un `binary long object`, c'est à dire un objet binaire long, qui peut contenir une certaine quantité d'information. Les quatre types TINYBLOB, BLOB, MEDIUMBLOB et LONGBLOB ne diffèrent que par leur taille maximum. Voir 7.2.1 Tailles nécessaires pour le stockage de types de colonnes.

Les quatre types TINYTEXT, TEXT, MEDIUMTEXT t LONGTEXT correspondent aux quatre types BLOB et ont les mêmes tailles maximum, et les mêmes conditions de stockage. La seule différence entre les types BLOB et TEXT tient aux tris et aux comparaisons : les tris et comparaisons tiennent compte de la casse, dans le cas des TEXT, et n'en tiennent pas compte, dans le cas des BLOB. En d'autres termes, un TEXT est un BLOB insensible à la casse.

Lors de l'affectation d'une valeur de type BLOB ou TEXT dans une colonne trop petite, la valeur est tronquée à la taille de la colonne.

En général, on peut considérer qu'une colonne de type TEXT est une colonne de type VARCHAR, aussi grande que désiré. De la même manière, on peut considérer qu'une colonne de type BLOB est une colonne de type VARCHAR BINARY, aussi grande que possible. Les différences sont :

- Il est possible d'indexer les variables de type BLOB et TEXT pour les version de ***MySQL*** 3.23.2 ou plus récentes. Les anciennes versions ne le supporte pas.
- Les espaces situés en début de chaîne ne sont pas effacés lors de l'enregistrement dans un BLOB t TEXT, contrairement aux colonnes de type VARCHAR. BLOB and TEXT columns cannot have DEFAULT values.
- Les valeurs de type BLOB t TEXT n'ont pas d'option DEFAULT.

MyODBC utilise le type LONGVARBINARY pour le type BLOB et LONGVARBINARY pour TEXT.

Parce que les types peuvent être extrêmement grands, il y a certaines contraintes à leur utilisation :

- Pour utiliser la clause GROUP BY ou ORDER BY sur une colonne de type BLOB ou TEXT, il faut commencer par convertir la colonne dans un type à longueur fixe. Pour cela, on utilise la fonction SUBSTRING. Par exemple :

```
mysql62; SELECT commentaires FROM Nom_table , SUBSTRING(commentaires,20) as
souschaîne ORDER BY souschaîne;
```

Si SUBSTRING n'est pas utilisé, le tri portera uniquement sur les `max_sort_longueur` premiers octets de la colonne. Par défaut, `max_sort_longueur` vaut 1024, et cette valeur peut être changée en utilisant l'option , lors du démarrage de ***MySQL***. Il est possible d'utiliser la clause GROUP sur un BLOB or TEXT , en spécifiant la position de la colonne, et en utilisant un alias. Par exemple :

```
mysql62; select id,substring(blob_col,1,100) FROM Nom_table
           GROUP BY 2;
mysql62; select id,substring(blob_col,1,100) as b FROM Nom_table
           GROUP BY b;
```

- La taille maximale d'un objet de type est déterminée par son type, mais la quantité d'information qu'il est possible de transmettre entre le serveur et le client dépend de la quantité de mémoire vive et de la taille des buffers de communications. Il est possible de changer la taille des buffers, mais il faut le faire sur le serveur et sur le client.

Il faut bien noter que chaque valeur de type est représentée de manière interne par un objet alloué. Contrairement aux autres objets, pour qui l'espace mémoire est réservé à la création de la table.

7.2.7.3 Le type ENUM

Le type est ENUM une chaîne, dont la valeur est choisi dans une liste de valeurs autorisées, et spécifiées à la création de la table.

Cette valeur peut prendre les valeurs ("") ou NULL sous certaines conditions :

- Lors de l'insertion d'une valeur invalide dans un ENUM (par exemple, une chaîne qui ne serait pas dans la liste des valeurs autorisées), une chaîne vide est insérée à la place, afin d'indiquer une erreur.
- Si une ENUM est déclarée NULL, NULL sera alors une valeur valide pour cette colonne, et la valeur par défaut sera aussi NULL. Si une ENUM est déclarée NOT NULL, la valeur par défaut sera le premier élément de la liste de valeurs autorisées.

Chaque énumération a un index.

- Les valeurs autorisées sont ordonnées et indexées à partir de 1.
- L'index de la valeur erreur est 0. Cela signifie qu'il est possible d'utiliser la clause SELECT pour rechercher des lignes qui auraient une valeur ENUM invalide :

```
mysql62; SELECT * FROM Nom_table WHERE enum_col=0;
```

- L'index de la valeur NULL est NULL.

Par exemple, une colonne de type ENUM("un", "deux", "trois") peut prendre chacune des valeurs ci−dessous. L'index de la valeur est aussi indiqué.

Value	*Index*
NULL	NULL
""	0
"one"	1
"two"	2
"three"	3

Une énumération peut avoir au maximum 65535 éléments.

La casse des lettres est sans importance pour l'affection de valeur dans une colonne de type ENUM. Cependant, lorsque ces valeurs sont retournées, elles tiennent compte de la casse des lettres tels qu'elle a été spécifiées à la création de la table.

Lire une valeur de type ENUM dans un contexte numérique permet d'accèder à l'index de la valeur. De même, lors de l'affectation d'un nombre dans une valeur ENUM, le nombre sera traité comme un index, et la valeur enregistrée sera celle de l'énumération, à l'index précisé.

Les valeurs d'une énumération sont triée en fonction de l'ordre dans lequel les éléments de l'énumération sont enregistrés lors de la création de la colonne (en d'autres termes, les valeurs d'une énumération sont triées en fonction de leur index). Par exemple, "a" sera placé avant "b" pour ENUM("a", "b"), mais "b" sera placé avant "a" pour ENUM("b", "a").les chaînes vides sont placées avant les chaînes non vides, et la valeur NULL passe avant toutes les autres.

Pour visualiser toutes les valeurs d'une colonne de type, il faut utiliser SHOW COLUMNS FROM Nom_table LIKE Nom_col_enum et détailler les valeurs de la deuxième colonne.

7.2.7.4 Le type SET

Un ensemble est une chaîne qui peut avoir aucune, une ou plusieurs valeurs, chaque valeur étant choisie dans une liste de valeur autorisées lors de la création de la table. Les valeurs de type SET qui ont plusieurs valeurs sont spécifiées en séparant les membres par des virgules(``,'').. Par conséquent, les valeurs de type SET ne peuvent pas contenir de virgule.

Par exemple, une colonne de type `SET("un", "deux") NOT NULL` peut prendre les valeurs suivantes :

```
""
"un"
"deux"
"un, deux"
```

Un ensemble peut avoir un maximum de 64 membres distincts.

MySQL enregistre les valeurs de type SET values numériquement, avec le premier élément situé au bit de poids faible. Lorsqu'une valeur de type SET est retournée dans un contexte numérique, les bits à 1 de cette valeurs correspondent à un membre du SET qui appartienne à cette valeur. Si un nombre est enregistré dans une valeur de type SET, alors les bits mis à un de ce nombre détermineront les membres du SET qui appartiennent à la valeur. Par exemple, une colonne a été spécifiée par `SET("a","b","c","d")`.Alors, les membres prennent la valeur suivante :

SET *member*	*Decimal value*	*Binary value*
a	1	0001
b	2	0010
c	4	0100
d	8	1000

Pour les valeurs qui contiennent plus d'un membre, l'ordre d'insertion n'importe pas. Une valeur peut être insérée plusieurs fois, elle n'apparaîtra plus qu'une seule fois dans le SET, et sera placé dans l'ordre des membres, à la création du SET. Par exemple, dans une colonne de type `SET("a","b","c","d")`, alors les valeurs `"a, d"`, `"d, a"` et `"d,a,a,d,d"` seront devenues `"a, d"`, lorsqu'elle seront retournées par la base.

Les valeurs de type s SET ont triées par ordre numérique. La valeur NULL est placée avant toutes les autres valeurs non− NULL.

En général, il est possible d'utiliser la clause SELECT sur une colonne de type SET, en utilisant l'opérateur LIKE ou la fonction `FIND_IN_SET()`.

```
mysql62; SELECT * FROM Nom_table WHERE Nom_col LIKE '%valeur%';
mysql62; SELECT * FROM Nom_table WHERE FIND_IN_SET(valeur, Nom_col)62;0;
```

Mais les exemples suivants sont aussi corrects

```
mysql62; SELECT * FROM Nom_table WHERE Nom_col = 'val1,val2';
mysql62; SELECT * FROM Nom_table WHERE Nom_col 1;
```

le premier exemple recherche une valeur exacte ('val1,val2'). Le second exemple recherche les valeurs qui contiennent le premier élément.

Pour visualiser toutes les valeurs d'une colonne de type, il faut utiliser `SHOW COLUMNS FROM Nom_table LIKE Nom_col_enum` et détailler les valeurs de la deuxième colonne.

7.2.8 Choisir le bon type de colonne

Pour une utilisation aussi efficace de l'espace mémoire, il faut utiliser le type de colonne le plus précis possible. Par exemple, pour stocker un entier dont la valeur va de `1` t `99999`, `MEDIUMINT UNSIGNED` est le meilleur type.

La représentation des valeurs monétaires est un problème commun. Avec ***MySQL***, le meilleur choix est le type `DECIMAL`. Il est enregistré comme une chaîne, ce qui n'entraîne aucune perte de données. Si la précision n'est pas primordiale, le type `DOUBLE` peut être un bon choix.

Pour une meilleure précision, il est toujours possible de convertir les nombres à virgule fixe en `BIGINT`. Cela autorise la manipulation d'entier pour les calculs, et il suffit alors de les reconvertir en valeur à virgule flottante au moment de l'affichage. 10.17 Quel sont les différents formats de lignes? Quand utiliser `VARCHAR/CHAR`?.

7.2.9 Index de colonne

Avec ***MySQL***, tous les types de colonnes peuvent être indexés, à l'exception des types `BLOB` et `TEXT`. L'utilisation d'index est le meilleur moyen d'accélérer les performances des clauses `SELECT`.

Une table peut avoir jusqu'à 16 index. La taille maximale d'un index est de 256 bytes, et cette valeur peut être choisie à la compilation de ***MySQL***.

Il n'est pas possible d'indexer une colonne qui contient des valeurs `NULL`, donc une colonne indexée doit être déclarée `NOT NULL`.

Pour les colonnes de type `CHAR` et `VARCHAR`, il est possible de préfixer la colonne. C'est un moyen beaucoup plus rapide et qui requiert moins d'espace disque qu'indexer une colonne complète. La syntaxe pour créer une telle colonne est la suivante :

```
KEY Nom_index(Nom_col(longueur))
```

L'exemple suivant créer un index pour les 10 premiers caractères de la colonne Nom_col.

```
mysql62; CREATE TABLE test (
           nom  CHAR(200) NOT NULL,
           KEY Nom_index(nom(10)));
```

7.2.10 Index multi−colonnes

MySQL peut créer des indexes sur plusieurs colonnes en même temps. Un index peut contenir jusqu'à 15 colonnes (Avec les colonnes de type `CHAR` t `VARCHAR` , il est aussi possible d'utiliser un préfixe lors de l'indexation).

Un index de plusieurs colonnes peut être considéré comme un tableau trié contenant les lignes obtenues en concaténant les valeurs des colonnes indexées.

MySQL gère les index sur plusieurs colonnes de manière à ce que les requêtes qui recherche une valeur connue dans la première colonne (avec une clause `WHERE`), soient rapides, même si les valeurs pour les autres colonnes ne sont pas précisées.

Par exemple, soit la table suivante :

```
mysql62; CREATE TABLE test (
           id INT NOT NULL,
           nom CHAR(30) NOT NULL,
           prenom CHAR(30) NOT NULL,
           PRIMARY KEY (id),
           INDEX nom_complet (nom, prenom));
```

Ainsi, l'index nom_complet est un index sur les deux colonnes nom et prénom. L'index sera utilisé lors des requêtes qui recherchent un nom, ou un nom et un prénom. L'index sera donc utilisé lors des requêtes suivantes :

```
mysql62; SELECT * FROM test WHERE nom="Dupont";
mysql62; SELECT * FROM test WHERE nom ="Dupont"
                          AND prenom="Michel";
mysql62; SELECT * FROM test WHERE nom ="Dupont"
                          AND (prenom ="Michel" OR prenom ="Marie");
mysql62; SELECT * FROM test WHERE nom ="Dupont"
                          AND prenom 62;="M" AND prenom < "N";
```

Cependant, l'index ne sera pas utilisé lors des requêtes suivantes :

```
mysql62; SELECT * FROM test WHERE prenom ="Michel";
mysql62; SELECT * FROM test WHERE nom="Dupont"
                          OR prenom ="Michel";
```

7.2.11 Utiliser des types de colonnes d'autres bases de données

Afin de simplifier le portage d'applications écrite en SQL sur d'autres base de données, ***MySQL*** remplace automatiquement les types présentés ci−dessous par les siens :

Other vendor type	*MySQL type*
BINARY(NUM)	CHAR(NUM) BINARY
CHAR VARYING(NUM)	VARCHAR(NUM)
FLOAT4	FLOAT
FLOAT8	DOUBLE
INT1	TINYINT
INT2	SMALLINT
INT3	MEDIUMINT
INT4	INT
INT8	BIGINT
LONG VARBINARY	MEDIUMBLOB
LONG VARCHAR	MEDIUMTEXT
MIDDLEINT	MEDIUMINT

VARBINARY(NUM)	VARCHAR(NUM) BINARY

Ce remplacement intervient à la création de la table. Après la création d'une table, les types retournés par une commande `DESCRIBE Nom_table` seront les types équivalent de ***MySQL***.

7.3 Fonctions utilisées dans les clauses SELECT et WHERE

Une clause `select` ou `where` dans une command SQL peut contenir des expressions utilisant les fonctions décrites ci−dessous.

Une expression qui contient une valeur aura toujours un résultat , sauf exceptions indiquées dans la documentation sur les opérateurs et fonctions.

Note: Il faut nécessairement un espace entre le nom de la fonction et la parenthèse ouvrante qui suit. Cela aide l'analyseur syntaxique de ***MySQL*** à faire la différence entre les appels de fonctions et les références aux tables ou colonnes qui interviennent dans la même colonne et qui ont le même nom que la fonction. Les espaces autours des arguments aussi autorisés.

Par souci de brièveté, les affichages d'exemples seront en mode réduit. Par exemple

```
mysql62; select MOD(29,9);
1 rows in set (0.00 sec)
+------------+
| mod(29,9) |
+------------+
|          2 |
+------------+
```

Sera affiché comme suit :

```
mysql62; select MOD(29,9);
        −62; 2
```

7.3.1 Fonction de groupement

```
( ... )
( ... )
```

Parenthèses. ***MySQL*** gère les priorités avec les parenthèses.

```
mysql62; select 1+2*3;
        −62; 7
mysql62; select (1+2)*3;
        −62; 9
```

Parentheses. Use these to force the order of evaluation in an expression.

7.3.2 Opérations arithmétiques normales

Les opérateurs arithmétiques usuels sont disponible. Il faut bien noter que dans le cas de `−`, `+` et `*`, le resultat est calculé avec la précision `BIGINT` (64−bit) si les deux opérateurs sont des entiers.

- + Addition

```
mysql62; select 3+5;
        −62; 8
```

- − Soustraction

```
mysql62; select 3−5;
        −62; −2
```

- * Multiplication

```
mysql62; select 3*5;
        −62; 15
mysql62; select 18014398509481984*18014398509481984.0;
        −62; 324518553658426726783156020576256.0
mysql62; select 18014398509481984*18014398509481984;
        −62; 0
```

Le résultat du dernier exemple est incorrect, car le résultat de la multiplication excède la taille maximale d'un `BIGINT`.

- / Division

```
mysql62; select 3/5;
        −62; 0.60
```

La division par zéro retourne une valeur `NULL`:

```
mysql62; select 102/(1−1);
        −62; NULL
```

La division sera calculée avec l'arithmétique des `BIGINT`, uniquement dans le cas ou le résultat doit être converti en entier !

7.3.3 Fonctions sur les bits

MySQL utilise des `BIGINT` (64−bit) pour ses opérations sur les bits, donc tous les opérateurs suivants ont au portent au plus 64 bits.

- | OU bit à bit

```
mysql62; select 29 | 15;
        −62; 31
```

- & ET bit à bit

```
mysql62; select 29 15;
        −62; 13
```

- << Décalage des bits vers la gauche sur un `BIGINT`.

```
mysql62; select 1 60;60; 2
        −62; 4
```

- >> Décalage des bits vers la droite sur un `BIGINT`.

```
mysql62; select 4 62;62; 2
        −62; 1
```

- `BIT_COUNT(N)` Compte le nombre de bits mis à un dans l'argument N.

```
mysql62; select BIT_COUNT(29);
        −62; 4
```

7.3.4 Opérations logiques

Toutes les opérations logiques retournent `1` (TRUE) ou `0` (FALSE).

- `NOT`
- `!` NON logique. Retourne 1 si l'argument est `0`, sinon, retourne `0`. Exception: `NOT NULL` retourne `NULL`.

```
mysql62; select NOT 1;
        −62; 0
mysql62; select NOT NULL;
        −62; NULL
mysql62; select ! (1+1);
        −62; 0
mysql62; select ! 1+1;
        −62; 1
```

Le dernier exemple retourne 1 car l'expression est évaluée de la même façon que `(!1)+1`.

- `OR`
- `||` OU logique. Retourne `1` si l'un des arguments est ni `0` ni `NULL`.

```
mysql62; select 1 || 0;
        −62; 1
mysql62; select 0 || 0;
        −62; 0
mysql62; select 1 || NULL;
        −62; 1
```

- `AND`
- `& ET logique. Retourne 0 si l'un des arguments est 0 ou NULL. Sinon, retourne 1.`

```
mysql62; select 1 #38; NULL;
        −62; 0
mysql62; select 1 #38; 0;
        −62; 0
```

7.3.5 Opérateurs de comparaison

Les opérations de comparaison retourne soit `1` (TRUE), `0` (FALSE) ou `NULL`. Ces fonctions s'appliquent aussi bien aux nombres qu'aux chaînes. Les chaînes sont automatiquement converties en nombre et les nombres en chaîne, suivant les besoins (comme en Perl).

MySQL effectue les comparaisons en suivant les règle suivantes

- Si l'un au moins des argument est `NULL`, le résultat de la comparaison est `NULL`.
- Si les deux argument sont des chaînes, ils se comparent en tant que chaînes.
- Si les deux argument sont des entiers, ils se comparent en tant que entiers.
- Si les deux argument est de type `TIMESTAMP` ou `DATETIME`, et l'autre argument est une constante, la constante est convertie au format `TIMESTAMP` avant la comparaison. Cette fonction est faite pour assurer la compatibilité ODBC.
- Dans tous les autres cas, les arguments sont comparés en tant que nombre à virgule flottante.

Par défaut, les comparaisons de chaîne sont fait sans tenir compte de la casse, et en utilisant le jeu de caractère courant (ISO−8859−1 Latin1 par défaut, qui fonctionne bien pour l'anglais). Les exemples suivants illustrent les règles de conversions lors des opérations de comparaison.

```
mysql62; SELECT 1 62; '6x';
         −62; 0
mysql62; SELECT 7 62; '6x';
         −62; 1
mysql62; SELECT 0 62; 'x6';
```

```
         −62; 0
mysql62; SELECT 0 = 'x6';
         −62; 1
```

- = Egalité.

```
mysql62; select 1 = 0;
        −62; 0
mysql62; select '0' = 0;
        −62; 1
mysql62; select '0.0' = 0;
        −62; 1
mysql62; select '0.01' = 0;
        −62; 0
mysql62; select '.01' = 0.01;
        −62; 1
```

- <>
- != Différent

```
mysql62; select '.01' 60;62; '0.01';
        −62; 1
mysql62; select .01 60;62; '0.01';
        −62; 0
mysql62; select 'zorro' 60;62; 'zorrro';
        −62; 1
```

- <= Inférieur ou égal

```
mysql62; select 0.1 60;= 2;
        −62; 1
```

- < Strictement inférieur

```
mysql62; select 2 60;= 2;
        −62; 1
```

- >= Supérieur ou égal

```
mysql62; select 2 62;= 2;
        −62; 1
```

- > Strictement supérieur

```
mysql62; select 2 62; 2;
        −62; 0
```

- <=> Egalité : ce opérateur s'assure qu'on ne compare pas NULL et une valeur non nulle.

```
mysql62; select 1 60;=62; 1, NULL 60;=62; NULL, 1 60;=62; NULL;
        −62; 1 1 0
```

- `expression BETWEEN minimum AND maximum` Si l'expression est comprise entre le `minimum et le maximum`, alors `BETWEEN` retourne `1`, sinon `0`. Cette fonction est équivalente à l'expression `(minimum <= expression AND expression <= maximum)` si tous les arguments sont du même type. Le premier argument (`expression`) détermine la manière dont la comparaison va être faite. Si `expression` est une chaîne, la comparaison sera de type chaîne, et insensible à la casse. Si `expression` est une chaîne de type binaire, la comparaison sera de type chaîne, et sensible à la casse. Si `expression` est un entier, la comparaison sera de type numérique. Si `expression` est un nombre à virgule flottante, la comparaison sera de type numérique, à virgule flottante.

```
mysql62; select 1 BETWEEN 2 AND 3;
        −62; 0
mysql62; select 'b' BETWEEN 'a' AND 'c';
        −62; 1
mysql62; select 2 BETWEEN 2 AND '3';
        −62; 1
mysql62; select 2 BETWEEN 2 AND 'x−3';
        −62; 0
```

- `expression IN (value,...)` Retourne `1` si `expression` est un membre de la liste `IN`, sinon retourne `0`.Si toutes les valeurs de la liste `IN` sont constantes, alors elles sont toutes converties au type de `expression`, et triées. La recherche dans la listes est alors faite avec une

recherche binaire. Cela signifie que la recherche est très rapide si la liste `IN` ne contient que des constantes. Si `expression` est une chaîne sensible à la casse, la comparaison avec les valeurs de `IN` en tiendra compte.

```
mysql62; select 2 IN (0,3,5,'wefwf');
        −62; 0
mysql62; select 'wefwf' IN (0,3,5,'wefwf');
        −62; 1
```

- `expression NOT IN (value,...)` Identique à `NOT (expression IN (value,...))`.
- `ISNULL(expression )` Si `expression` est `NULL`, `ISNULL()` retourne `1`, sinon `0`.

```
mysql62; select ISNULL(1+1);
        −62; 0
mysql62; select ISNULL(1/0);
        −62; 1
```

Il faut noter que la comparaison à `NULL` avec = sera toujours fausse!

- `COALESCE(liste)` Retourne le premier élément non `NULL` dans la liste.

```
mysql62; select COALESCE(NULL,1);
        −62; 1
mysql62; select ISNULL(NULL,NULL,NULL);
        −62; NULL
```

- `INTERVAL(N,N1,N2,N3,...)` Retourne `1` si `N1` < `N2` < `N3` < `...` < `Nn` . Si il existe deux valeurs i et j, telles que i < j et Ni > Nj, alors la fonction retourne faux. Toutes les valeurs sont traitées comme des nombres. Cela permet l'utilisation d'une comparaison binaire, très rapide.

```
mysql62; select INTERVAL(23, 1, 15, 17, 30, 44, 200);
        −62; 3
mysql62; select INTERVAL(10, 1, 10, 100, 1000);
        −62; 2
mysql62; select INTERVAL(22, 23, 30, 44, 200);
        −62; 0
```

7.3.6 Fonctions de comparaisons des chaînes

Dans une expression quelconque, si une des chaînes est sensible à la casse, alors la comparaison tiendra compte de la casse.

- `expr1 LIKE expr2 [ESCAPE 'escape−char']` Expression régulière SQL. Retourne `1` (TRUE) ou `0` (FALSE). Avec `LIKE`, les caractères spéciaux suivants sont disponibles :

```
mysql62; select 'David!' LIKE 'David_';
        −62; 1
mysql62; select 'David!' LIKE '%D%v%';
        −62; 1
```

Pour tester la présence d'un des caractères spéciaux, il suffit de faire précéder du caractère d'échappement. Si ce dernier n'est pas précisé, le caractère ``\'' est alors utilisé.

```
mysql62; select 'David!' LIKE 'David\_';
        −62; 0
mysql62; select 'David_' LIKE 'David\_';
        −62; 1
```

Pour spécifier un autre caractère d'échappement, il faut utiliser la clause ESCAPE :

```
mysql62; select 'David_' LIKE 'David|_' ESCAPE '|';
       −62; 1
```

`LIKE` peut travailler avec des expressions numériques (c'est une extension de la norme ANSI SQL.)

```
mysql62; select 10 LIKE '1%';
        −62; 1
```

Note: Comme ***MySQL*** utilise le système d'échappement du langage C (e.g., ``\n''), il faut doubler toutes les occurrences dans les clause LIKE. Par exemple, pour rechercher le caractère ``\n'', il faut mettre la chaîne ``\\n''. Pour recherche le caractère for ``\'', il faut l'écrire ``\\\\'' (les backslashes sont supprimés une première fois par l'analyseur syntaxique, et une deuxième fois, quand la recherche est terminée : ce qui laisse un seul backslash à rechercher).

- `expr1 NOT LIKE expr2 [ESCAPE 'escape−char']` Identique à `NOT (expr1 LIKE expr2 [ESCAPE 'escape−char'])`.
- expr REGEXP pat
- expr RLIKE pat
- `expression REGEXP pat`
- `expression RLIKE pat` Effectue une chercher sur la chaîne `expression` avec le masque `pat`. Le pattern peut être une expression régulière étendue. Retoune `1` si l'expression régulière réussit, `0`. `RLIKE` est un synonyme pour `REGEXP`, fourni pour assurer la compatibilité avec mSQL Note: Comme ***MySQL*** utilise le système d'échappement du langage C (e.g., ``\n''), il faut doubler toutes les occurrences dans les clause. `REGEXP` .

```
mysql62; select 'Monty!' REGEXP 'm%y%%';
        −62; 0
mysql62; select 'Monty!' REGEXP '.*';
        −62; 1
mysql62; select 'new*\n*line' REGEXP 'new\\*.\\*line';
        −62; 1
```

REGEXP et RLIKE utilisent le jeu de caractère (ISO−8859−1 Latin1 par défaut) pour choisir le type de caractère.

- expr NOT REGEXP pat
- expr NOT RLIKE pat
- `expression NOT REGEXP expression` Identique à `NOT (expression REGEXP expression )`.
- `STRCMP(expr1,expr2) STRCMP()` retourne `0` si les chaînes sont identiques, `−1` si le premier argument est plus petit que le second, en fonction de l'ordre de tri courant, et sinon `1` .

```
mysql62; select STRCMP('text', 'text2');
        −62; −1
mysql62; select STRCMP('text2', 'text');
        −62; 1
mysql62; select STRCMP('text', 'text');
        −62; 0
```

7.3.7 Opérateurs de transtypage

- `BINARY` L'opérateur `BINARY` transforme la chaîne qui le suive en chaîne insensible à la casse. C'est un moyen simple de forcer une comparaison insensible à la casse, même si la colonne n'a pas été définie comme un `BINARY` ou `BLOB`.

```
mysql62; select "a" = "A";
        −62; 1
mysql62; select BINARY "a" = "A";
        −62; 0
```

BINARY a été introduit dans ***MySQL*** 3.23.0

7.3.8 Fonctions de contrôle

- `IFNULL(expr1,expr2)` Si `expr1` n'est pas `NULL`, `IFNULL()` retourne `expr1`, sinon `expr2`. `IFNULL()` retourne un nombre ou une chaîne, en fonction du contexte dans lequel il est utilisé.

```
mysql62; select IFNULL(1,0);
```

```
        −62; 1
mysql62; select IFNULL(0,10);
        −62; 0
mysql62; select IFNULL(1/0,10);
        −62; 10
mysql62; select IFNULL(1/0,'yes');
        −62; 'yes'
```

- `IF(expr1,expr2,expr3)` Si `expr1` est vraie (TRUE) (`expr1 <> 0` et `expr1 <> NULL`) alors `IF()` retourne `expr2`, sinon il retourne `expr3`. `IF()` retourne un nombre ou une chaîne, en fonction du contexte dans lequel il est utilisé.

```
mysql62; select IF(162;2,2,3);
        −62; 3
mysql62; select IF(160;2,'yes','no');
        −62; 'yes'
mysql62; select IF(strcmp('test','test1'),'yes','no');
        −62; 'no'
```

`expr1` est évaluée en tant qu'entier, ce qui signifie que si le test portait sur un nombre à virgule flottante ou une chaîne, il faudrait mieux utiliser un opérateur de comparaison.

```
mysql62; select IF(0.1,1,0);
        −62; 0
mysql62; select IF(0.160;62;0,1,0);
        −62; 1
```

Dans le premier cas ci−dessus, `IF(0.1)` retourne `0` car `0.1` est converti en un nombre entier, ce qui conduit à un test de type `IF(0)`. C'est une source d'erreur. Dans le deuxième cas, la comparaison est faite avec un opérateur de comparaison, qui teste la non nullité. Le résultat de la comparaison est utilisé comme un entier.

- CASE value WHEN [compare−value] THEN result [WHEN [compare−value] THEN result ...] [ELSE result] END
- CASE WHEN [condition] THEN result [WHEN [condition] THEN result ...] [ELSE result] END La première version retourne `result` si `value=compare−value`. La seconde version retourne le résultat si la première condition est vraie. Si il n'y a aucun résultat, alors, le résultat après `ELSE` est retourné. SI il n'y a pas de résultat pour `ELSE` alors `NULL`.

```
mysql62; SELECT CASE 1 WHEN 1 THEN "un" WHEN 2 THEN "deux" ELSE "plus" END;
       −62; "un"
mysql62; SELECT CASE WHEN 162;0 THEN "vrai" ELSE "faux" END;
       −62; "vrai"
mysql62; SELECT CASE BINARY "B" when "a" then 1 when "b" then 2 END;
       −62; NULL
```

7.3.9 Fonctions mathématiques

Toutes les fonctions mathématiques retourne `NULL` en cas d'erreur.

- − Moins unaire. Change le signe de l'argument.

```
mysql62; select − 2;
        −62; −2
```

Note : si cet opérateur est utilisé avec un `BIGINT`, le résultat sera `BIGINT`! Cela signifie qu'il faut éviter d'utiliser − sur des entiers qui ont la valeur `−2^63`!

- `ABS(X)` Valeur absolue de `X`.

```
mysql62; select ABS(2);
        −62; 2
mysql62; select ABS(−32);
```

```
        −62; 32
```

Cette fonction ne pose aucun problème particulier avec les valeurs de type BIGINT.

- SIGN(X) Retourne le signe de l'argument sous la forme −1, 0 or 1, suivant que X est négatif, nul, ou positif.

```
mysql62; select SIGN(−32);
        −62; −1
mysql62; select SIGN(0);
        −62; 0
mysql62; select SIGN(234);
        −62; 1
```

- MOD(N,M)
- % Modulo (identique à l'opérateur % en langage C). Retourne le reste de la division euclidienne de N par M.

```
mysql62; select MOD(234, 10);
        −62; 4
mysql62; select 253 % 7;
        −62; 1
mysql62; select MOD(29,9);
        −62; 2
```

Cette fonction ne pose aucun problème particulier avec les valeurs de type BIGINT.

- FLOOR(X) Retourne le plus grand entier possible mais plus petit que X.

```
mysql62; select FLOOR(1.23);
        −62; 1
mysql62; select FLOOR(−1.23);
        −62; −2
```

Note : le résultat est converti en BIGINT!

- CEILING(X) Retourne le plus petit entier possible mais plus grand que X.

```
mysql62; select CEILING(1.23);
        −62; 2
mysql62; select CEILING(−1.23);
        −62; −1
```

Note : le résultat est converti en BIGINT!

- ROUND(X) Retourne l'argument X, arrondi à l'entier le plus proche.

```
mysql62; select ROUND(−1.23);
        −62; −1
mysql62; select ROUND(−1.58);
        −62; −2
mysql62; select ROUND(1.58);
        −62; 2
```

Note : le résultat est converti en BIGINT!

- ROUND(X,D) Retourne l'argument X, arrondi au décimal le plus proche, avec D décimales. Si D =0, le résultat n'aura pas de partie décimale.

```
mysql62; select ROUND(1.298, 1);
        −62; 1.3
mysql62; select ROUND(1.298, 0);
        −62; 1
```

Note : le résultat est converti en BIGINT!

- EXP(X) Retourne la valeur de e (base des logarithmes naturels ou népériens) à la puissance X.

```
mysql62; select EXP(2);
```

```
        −62; 7.389056
mysql62; select EXP(−2);
        −62; 0.135335
```

- `LOG(X)` Retourne le logarithme naturel de `X`.

```
mysql62; select LOG(2);
        −62; 0.693147
mysql62; select LOG(−2);
        −62; NULL
```

Pour obtenir la valeur du logarithme de `X` dans une base arbitraire, il faut utiliser la formule `LOG(X)/LOG(B)`.

- `LOG10(X)` Retourne le logarithme de `X` en base 10.

```
mysql62; select LOG10(2);
        −62; 0.301030
mysql62; select LOG10(100);
        −62; 2.000000
mysql62; select LOG10(−100);
        −62; NULL
```

- `POW(X,Y)`
- `POWER(X,Y)` Retourne la valeur de `X` à la puissance `Y`.

```
mysql62; select POW(2,2);
        −62; 4.000000
mysql62; select POW(2,−2);
        −62; 0.250000
```

- `SQRT(X)` Retourne la racine carrée positive de `X`.

```
mysql62; select SQRT(4);
        −62; 2.000000
mysql62; select SQRT(20);
        −62; 4.472136
```

- `PI()` Retourne la valeur de PI.

```
mysql62; select PI();
        −62; 3.141593
```

- `COS(X)` Retourne le cosinus de `X`, avec `X` en radians.

```
mysql62; select COS(PI());
        −62; −1.000000
```

- `SIN(X)` Retourne le sinus de `X`, avec `X` en radians.

```
mysql62; select SIN(PI());
        −62; 0.000000
```

- `TAN(X)` Retourne la tangente de `X`, avec `X` en radians.

```
mysql62; select TAN(PI()+1);
        −62; 1.557408
```

- `ACOS(X)` Retourne l'arcosinus de `X`, c'est à dire l'angle dont le cosinus est `X` en radians. Retourne `NULL` si `X` n'est pas compris entre `−1` et `1`.

```
mysql62; select ACOS(1);
        −62; 0.000000
mysql62; select ACOS(1.0001);
        −62; NULL
mysql62; select ACOS(0);
        −62; 1.570796
```

- `ASIN(X)` Retourne l'arsinus de `X`, c'est à dire l'angle dont le sinus est `X` en radians. Retourne `NULL` si `X` n'est pas compris entre `−1` et `1`.

```
mysql62; select ASIN(0.2);
        −62; 0.201358
mysql62; select ASIN('foo');
        −62; 0.000000
```

- `ATAN(X)` Retourne l'arctangente de `X`, c'est à dire l'angle dont la tangente est `X` en radians.

```
mysql62; select ATAN(2);
        −62; 1.107149
mysql62; select ATAN(−2);
        −62; −1.107149
```

- `ATAN2(X,Y)` Retourne l'arc tangente de deux variables `X` et `Y`. C'est le même calcul que arc tangent of `Y / X`, hormis le fait que les signes des deux arguments est utilisé pour déterminer le quadrant du résultat.

```
mysql62; select ATAN(−2,2);
        −62; −0.785398
mysql62; select ATAN(PI(),0);
        −62; 1.570796
```

- `COT(X)` Retourne la cotangente de `X`.

```
mysql62; select COT(12);
        −62; −1.57267341
mysql62; select COT(0);
        −62; NULL
```

- `RAND()`
- `RAND(N)` Retourne un nombre aléatoire, compris entre `0` et `1.0`. Si un entier `N` est précisé, il est utilisé comme valeur de seed.

```
mysql62; select RAND();
        −62; 0.5925
mysql62; select RAND(20);
        −62; 0.1811
mysql62; select RAND(20);
        −62; 0.1811
mysql62; select RAND();
        −62; 0.2079
mysql62; select RAND();
        −62; 0.7888
```

Il est impossible d'utiliser une colonne de valeur `RAND()` avec la clause `ORDER BY` , car la colonne sera évalué plusieurs fois. Avec ***MySQL*** 3.23, il est cependant possible d'écrire: `SELECT * FROM Nom_table ORDER BY RAND()`. Comme cela, il est possible de faire une sélection aléatoire d'une table : `SELECT * FROM table1,table2 WHERE a=b AND c<d ORDER BY RAND() LIMIT 1000`.

- `LEAST(X,Y,...)` Au moins deux arguments, retourne la plus petite valeur (minimum). Les arguments sont comparés en utilisant les règles suivantes :
  - ♦ Si `LEAST` est utilisé dans un contexte d'entiers, ou bien tous les arguments sont des entiers, les arguments sont évalués et comparés en tant qu'entiers.
  - ♦ Si `LEAST` est utilisé dans un contexte de nombre à virgule flottante, les arguments sont évalués et comparés en tant que nombre à virgule flottante.
  - ♦ Si tous les arguments sont des chaînes sensibles à la casse, tous les arguments sont comparés comme des chaînes insensibles à la casse.
  - ♦ Dans tous les autres cas, les arguments sont comparés comme des chaînes insensibles à la casse.

```
mysql62; select LEAST(2,0);
        −62; 0
mysql62; select LEAST(34.0,3.0,5.0,767.0);
        −62; 3.0
mysql62; select LEAST("B","A","C");
        −62; "A"
```

Avec les version de ***MySQL*** antérieur à la 3.22.5, il est possible d'utiliser `MIN()` à la place de `LEAST`.

- `GREATEST(X,Y,...)` Retourne le plus grand argument de la liste. Les arguments sont comparés de la même manière que pour `LEAST`.

```
mysql62; select GREATEST(2,0);
        −62; 2
mysql62; select GREATEST(34.0,3.0,5.0,767.0);
        −62; 767.0
mysql62; select GREATEST("B","A","C");
        −62; "C"
```

Avec les version de ***MySQL*** antérieur à la 3.22.5, il est possible d'utiliser `MAX ()` à la place de `GREATEST`.

- `DEGREES(X)` Retourne l'argument `X`, converti de radians en degrés.

```
mysql62; select DEGREES(PI());
        −62; 180.000000
```

- `RADIANS(X)` Retourne l'argument `X`, converti de radians degrés en radians.

```
mysql62; select RADIANS(90);
        −62; 1.570796
```

- `TRUNCATE(X,D)` Retourne l'argument `X`, tronqué à `D` décimales.

```
mysql62; select TRUNCATE(1.223,1);
        −62; 1.2
mysql62; select TRUNCATE(1.999,1);
        −62; 1.9
mysql62; select TRUNCATE(1.999,0);
        −62; 1
```

7.3.10 Fonctions de chaînes

Les fonctions qui retourne des chaînes, retourneront `NULL` si le résultat dépasse la taille maximale de `max_allowed_packet`.

Pour les opérations sur les chaînes, le premier caractère est en position 1.

- `ASCII(str)` Retourne le code ASCII du premier caractère de la chaîne. Si la chaîne est vide, retourne `0`. Si la chaîne est `NULL`, retourne `NULL`.

```
mysql62; select ASCII('2');
        −62; 50
mysql62; select ASCII(2);
        −62; 50
mysql62; select ASCII('dx');
        −62; 100
```

- `CONV(N,from_base,to_base)` Converti des nombres d'une base à l'autre, et retourne la chaîne représentant le résultat converti. Si la chaîne est `NULL`, retourne `NULL`. L'argument `N` est considéré comme un entier, mais peut être un entier ou une chaîne. La base minimum est 2, et maximum est 36. Si la base d'arrivée est un nombre négatif, `N` est considéré comme un entier signé. Sinon, `N` est traité comme un entiers non signé. `CONV` fonctionne à la précision 64 bits.

```
mysql62; select CONV("a",16,2);
        −62; '1010'
mysql62; select CONV("6E",18,8);
        −62; '172'
mysql62; select CONV(−17,10,−18);
        −62; '−H'
mysql62; select CONV(10+"10"+'10'+0xa,10,10);
        −62; '40'
```

- `BIN(N)` Retourne une chaîne représentant l'argument `N` en binaire. `N` est un `BIGINT`. Cette fonction est l'équivalent de `CONV(N,10,2)`. Retourne `NULL` si `N` est `NULL`.

```
mysql62; select BIN(12);
        −62; '1100'
```

- `OCT(N)` Retourne une chaîne représentant l'argument `N` en base 8. `N` est un `BIGINT`. Cette fonction est l'équivalent de `CONV(N,10,8)`. Retourne `NULL` si `N` est `NULL`.

```
mysql62; select OCT(12);
        −62; '14'
```

- `HEX(N)` Retourne une chaîne représentant l'argument `N` en hexadécimal. `N` est un `BIGINT`. Cette fonction est l'équivalent de `CONV(N,10,16)`. Retourne `NULL` si `N` est `NULL`.

```
mysql62; select HEX(255);
        −62; 'FF'
```

- `CHAR(N,...)` Interprète les arguments comme des nombres entiers, et retourne une chaîne constituée des caractères correspondant aux codes ASCII des arguments. Les valeurs `NULL` sont ignorées.

```
mysql62; select CHAR(77,121,83,81,'76');
        −62; 'MySQL'
mysql62; select CHAR(77,77.3,'77.3');
        −62; 'MMM'
```

- `CONCAT(X,Y,...)` Concatène les arguments, et re tourne le résultat. Retourne `NULL` si un des arguments est `NULL`. Le nombre d'argument minimum est 2.

```
mysql62; select CONCAT('My', 'S', 'QL');
        −62; 'MySQL'
mysql62; select CONCAT('My', NULL, 'QL');
        −62; NULL
```

- `LONGUEUR(str)`
- `OCTET_LONGUEUR(str)`
- `CHAR_LONGUEUR(str)`
- `CHARACTER_LONGUEUR(str)` Retourne la longueur de la chaîne `str`.

```
mysql62; select LONGUEUR('text');
        −62; 4
mysql62; select OCTET_LONGUEUR('text');
        −62; 4
```

- `LOCATE(substr,str)`
- `POSITION(substr IN str)` Retourne la position de la première occurrence de `substr` dans la chaîne. Retourne `0` si `substr` n'est pas trouvée.

```
mysql62; select LOCATE('bar', 'foobarbar');
        −62; 4
mysql62; select LOCATE('xbar', 'foobar');
        −62; 0
```

- `LOCATE(substr,str,pos)` Retourne la position de la première occurrence de `substr` dans la chaîne, en commencant à chercher à partir de la position `pos`. Retourne `0` si `substr` n'est pas trouvée.

```
mysql62; select LOCATE('bar', 'foobarbar',5);
        −62; 7
```

- `INSTR(str,substr)` Retourne la position de la première occurrence de `substr` dans la chaîne, en commencant à chercher à partir de la position `pos`. Retourne `0` si `substr` n'est pas trouvée. C'est la même fonction que `LOCATE()`, mais les deux arguments n'ont pas la même place.

```
mysql62; select INSTR('foobarbar', 'bar');
        −62; 4
mysql62; select INSTR('xbar', 'foobar');
        −62; 0
```

- `LPAD(str,len,padstr)` Retourne la chaîne `str`, complétée à gauche par la chaîne `padstr` jusqu'à ce que le résultat ait la longueur .

```
mysql62; select LPAD('hi',4,'??');
        −62; '??hi'
```

- `RPAD(str,len,padstr)` Retourne la chaîne `str`, complétée à droite par la chaîne `padstr` jusqu'à ce que le résultat ait la longueur .

```
mysql62; select RPAD('hi',5,'?');
        −62; 'hi???'
```

- `LEFT(str,len)` Retourne les `len` premiers caractères de la chaîne `str`.

```
mysql62; select LEFT('foobarbar', 5);
        −62; 'fooba'
```

- `RIGHT(str,len)`
- `SUBSTRING(str FROM len)` Retourne les `len` derniers caractères de la chaîne `str`.

```
mysql62; select RIGHT('foobarbar', 4);
        −62; 'rbar'
mysql62; select SUBSTRING('foobarbar' FROM 4);
        −62; 'rbar'
```

- `SUBSTRING(str,pos,len)`
- `SUBSTRING(str FROM pos FOR len)`
- `MID(str,pos,len)` Retourne les `len` caractères de la chaîne `str, en commencant à partir de la position pos`. La variante `FROM` est une syntaxe issue de la norme ANSI SQL92.

```
mysql62; select SUBSTRING('Quadratically',5,6);
        −62; 'ratica'
```

- `SUBSTRING(str,pos)` Retourne une sous−chaîne, issue de `str` et commencant à la position `pos`.

```
mysql62; select SUBSTRING('Quadratically',5);
        −62; 'ratically'
```

- `SUBSTRING_INDEX(str,delim,count)` Retourne une sous−chaîne, issue de `str`, après `count` occurrences du délimiteur `delim`. Si `count` est positif, la sous−chaîne comprendra tous les caractères situés à gauche du délimiteur final. Si `count` est négatif, la sous−chaîne comprendra tous les caractères situés à droite du délimiteur final

```
mysql62; select SUBSTRING_INDEX('www.mysql.com', '.', 2);
        −62; 'www.mysql'
mysql62; select SUBSTRING_INDEX('www.mysql.com', '.', −2);
        −62; 'mysql.com'
```

- `LTRIM(str)` Retourne la chaîne `str` après élimination des espaces situés en début de chaîne.

```
mysql62; select LTRIM('  barbar');
        −62; 'barbar'
```

- `RTRIM(str)` retourne la chaîne `str` après élimination des espaces situés en fin de chaîne.

```
mysql62; select RTRIM('barbar   ');
        −62; 'barbar'
```

- `TRIM([[BOTH | LEADING | TRAILING] [remstr] FROM] str)` Retourne la chaîne `str` après élimination de chaînes `remstr` situées en début et/ou fin de chaîne. Si aucune des options `BOTH`, `LEADING` ou `TRAILING` n'est précisé, alors `BOTH` est utilisé par défaut. Si `remstr` n'est pas précisé, les espaces sont éliminés.

```
mysql62; select TRIM('  bar   ');
        −62; 'bar'
mysql62; select TRIM(LEADING 'x' FROM 'xxxbarxxx');
        −62; 'barxxx'
mysql62; select TRIM(BOTH 'x' FROM 'xxxbarxxx');
        −62; 'bar'
mysql62; select TRIM(TRAILING 'xyz' FROM 'barxxyz');
        −62; 'barx'
```

- `SOUNDEX(str)` Retourne une représentation phonétique de la chaîne . Deux chaînes qui " sonne de la même façon " devraient avoir des représentation identiques. Une représentation standard a 4 caractères, mais `SOUNDEX()` retourne un nombre arbitraire de caractère. Il faudra alors utiliser `SUBSTRING()` sur le résultat pour avoir une représentation standard. Les caractères non−alphanumériques sont ignorés. Tous les caractères alphabétiques hors de l'intervalle A−Z sont considéré comme des voyelles.
- Attention : cette fonction ayant été programmé par des anglophones, la " sonorité " d'une chaîne aura un fort accent anglais.

```
mysql62; select SOUNDEX('Hello');
        −62; 'H400'
mysql62; select SOUNDEX('Quadratically');
        −62; 'Q36324'
```

- `SPACE(N)` Crée une chaîne contenant `N` espaces.

```
mysql62; select SPACE(6);
```

```
        −62; '      '
```

- `REPLACE(str,from_str,to_str)` Remplace les occurrences de `from_str` par la chaîne `to_str`, dans la chaîne `str`.

```
mysql62; select REPLACE('www.mysql.com', 'w', 'Ww');
        −62; 'WwWwww.mysql.com'
```

- `REPEAT(str,count)` Retourne une chaîne constituée des répétitions de la chaîne . Si `count <= 0`, retourne une chaîne vide. Si `str` ou `count` est `NULL`, retourne `NULL`.

```
mysql62; select REPEAT('MySQL', 3);
        −62; 'MySQLMySQLMySQL'
```

- `REVERSE(str)` Inverse l'ordre des caractères de la chaîne `str`.

```
mysql62; select REVERSE('abc');
        −62; 'cba'
```

- `INSERT(str,pos,len,newstr)` Retourne la chaîne `str`, avec la chaîne `newstr` qui remplace tous les caractères à partir de la position `pos`, et sur la longueur `len`.

```
mysql62; select INSERT('Quadratic', 3, 4, 'What');
        −62; 'QuWhattic'
```

- `ELT(N,str1,str2,str3,...)` Retourne `str1` si `N = 1`, `str2` si `N = 2`, etc... Retourne `NULL` si est plus petit que `1` ou plus grand que le nombre d'arguments. `ELT()` est le contraire de `FIELD()`.

```
mysql62; select ELT(1, 'ej', 'Heja', 'hej', 'foo');
        −62; 'ej'
mysql62; select ELT(4, 'ej', 'Heja', 'hej', 'foo');
        −62; 'foo'
```

- `FIELD(str,str1,str2,str3,...)` Retourne l'index de la chaîne `str` dans la liste des arguments . Retourne `0` si `str` n'est pas trouvé .`FIELD()` est le contraire de `ELT()`

```
mysql62; select FIELD('ej', 'Hej', 'ej', 'Heja', 'hej', 'foo');
        −62; 2
mysql62; select FIELD('fo', 'Hej', 'ej', 'Heja', 'hej', 'foo');
        −62; 0
```

- `FIND_IN_SET(str,strlist)` Retourne l'index de la chaîne dans la liste. Une liste de chaîne est composée de chaînes, séparées par le caractère ``,''. Si le premier argument est une chaîne constante, et le deuxième est une colonne de type `SET`, la fonction est optimisée `FIND_IN_SET()` pour utiliser l'arithmétique sur les bits ! Retourne `str` si n'est pas dans la liste `strlist`, ou si la liste est vide. Retourne `NULL` si l'un des arguments est `NULL`. La fonction n'accepte pas de caractère ``,''dans le premier membre de la liste.

```
mysql62; SELECT FIND_IN_SET('b','a,b,c,d');
        −62; 2
```

- `MAKE_SET(bits,str1,str2,...)` Retourne un ensemble (une chaîne contenant des sous−chaînes séparée par ``,'') qui correspond à u sous−ensemble des chaînes en arguments. La chaîne à l'index i sera présente dans l'ensemble résultat, si le bit à l'index i est à un, dans `bits).`

```
mysql62; SELECT MAKE_SET(1,'a','b','c');
        −62; 'a'
mysql62; SELECT MAKE_SET(1 | 4,'hello','nice','world');
        −62; 'hello,world'
mysql62; SELECT MAKE_SET(0,'a','b','c');
        −62; ''
```

- `EXPORT_SET(bits,on,off,[separator],[number_of_bits])` Retourne une chaîne dans laquelle, pour chaque bit à 1 , il y a la chaîne `on`, pour chaque bit à 0 , il y a la chaîne `off`, séparés par le `separator` (par défaut, ','), et uniquement pour les `number_of_bits` (par défaut, 64) premiers bits.

```
mysql62; select EXPORT_SET(5,'Y','N',',',4)
        −62; Y,N,Y,N
```

- `LCASE(str)`
- `LOWER(str)` Retourne la chaîne avec tous les caractères en minuscules, conformément au jeu de caractère courant (par défaut, ISO−8859−1 Latin1).

```
mysql62; select LCASE('QUADRATICALLY');
        −62; 'quadratically'
```

- `UCASE(str)`

- `UPPER(str)` Retourne la chaîne avec tous les caractères en majuscule, conformément au jeu de caractère courant (par défaut, ISO−8859−1 Latin1).
```
mysql62; select UCASE('Hej');
        −62; 'HEJ'
```

- `LOAD_FILE(Nom_fichier)` Lit le fichier `Nom_fichier` et retourne le résultat dans une chaîne. Le fichier doit être sur le serveur, et il faut préciser le nom et le chemin d'accès complet. Le fichier doit être lisible par tous, et être plus petit que `max_allowed_packet`. Si le fichier n'existe pas, ou ne peut être lu, la fonction retourne `NULL`.
```
mysql62; UPDATE Nom_tableSET blob_column=LOAD_FILE("/tmp/picture") WHERE id=1;
```

Il n'y a pas de fonction de conversion d'un nombre en char. Il n'y en pas besoin, car ***MySQL*** converti automatiquement les nombres en chaînes, et vice versa.

```
mysql62; SELECT 1+"1";
        −62; 2
mysql62; SELECT CONCAT(2,' test');
        −62; '2 test'
```

Si une fonction qui travaille sur des chaînes sensibles à la casse recoit comme argument une chaîne insensible à la casse, le resultat sera aussi une chaîne insensible à la casse. Un nombre converti en chaîne est traité comme une chaîne insensible à la casse. Cela n'affecte que les comparaisons.

7.3.11 Fonctions de date et heure

7.2.6 Types date et heure pour une description précise des intervalles de validité de chaque type.

Voici un exemple qui utilise des fonctions sur les dates et heures. La requête ci−dessous sélectionne toutes les lignes avec une valeur qui est `date_col` dans les 30 derniers jours :

```
mysql62; SELECT quelquechose FROM table
           WHERE TO_DAYS(NOW()) − TO_DAYS(date_col) 60;= 30;
```

- `DAYOFWEEK(date)` Retourne le jour de la semaine sous la forme d'un index qui commence à 1 (`1` = Dimanche, `2` = Lundi, ... `7` = Samedi). Ces valeurs sont celles du standard ODBC.
```
mysql62; select DAYOFWEEK('1998−02−03');
        −62; 3
```

- `WEEKDAY(date)` Retourne le jour de la semaine sous la forme d'un index qui commence à 0 (`0` = Dimanche, `1` = Lundi, ... `6` = Samedi).
```
mysql62; select WEEKDAY('1997−10−04 22:23:00');
        −62; 5
mysql62; select WEEKDAY('1997−11−05');
        −62; 2
```

- `DAYOFMONTH(date)` Retourne le jour du mois sous la forme d'un index entre `1` et `31`.
```
mysql62; select DAYOFMONTH('1998−02−03');
        −62; 3
```

- `DAYOFYEAR(date)` Retourne le jour de l'année sous la forme d'un index entre `1` et `366`.
```
mysql62; select DAYOFYEAR('1998−02−03');
        −62; 34
```

- `MONTH(date)` Retourne le mois de la `date` sous la forme d'un index entre `1` et `12`.
```
mysql62; select MONTH('1998−02−03');
        −62; 2
```

- `DAYNAME(date)` Retourne le nom du jour de la `date` sous la forme d'une chaîne (en anglais).
```
mysql62; select DAYNAME("1998−02−05");
        −62; 'Thursday'
```

- `MONTHNAME(date)` Retourne le nom du mois de la `date` sous la forme d'une chaîne (en anglais).

```
mysql62; select MONTHNAME("1998−02−05");
        −62; 'February'
```

- `QUARTER(date)` Retourne le trimestre de la `date` sous la forme d'un index entre `1` et `4`.

```
mysql62; select QUARTER('98−04−01');
        −62; 2
```

- `WEEK(date)`
- `WEEK(date,first)` Avec un seul argument, retourne la semaine de la `date` sous la forme d'un index entre `1` à `52`, avec Dimanche comme premier jour de la semaine. La fonction avec deux arguments permet de préciser si la semaine commence à Dimanche (`0`) ou Lundi (`1`).

```
mysql62; select WEEK('1998−02−20');
        −62; 7
mysql62; select WEEK('1998−02−20',0);
        −62; 7
mysql62; select WEEK('1998−02−20',1);
        −62; 8
```

- `YEAR(date)` Retourne l'année de la `date` sous la forme d'un index entre `1000` et `9999`.

```
mysql62; select YEAR('98−02−03');
        −62; 1998
```

- `HOUR(time)` Retourne l'heure de la `date` sous la forme d'un index entre `0` et `23`.

```
mysql62; select HOUR('10:05:03');
        −62; 10
```

- `MINUTE(time)` Retourne la minute de la `date` sous la forme d'un index entre `0` et `59`.

```
mysql62; select MINUTE('98−02−03 10:05:03');
        −62; 5
```

- `SECOND(time)` Retourne la seconde de la `date` sous la forme d'un index entre `0` et `59`.

```
mysql62; select SECOND('10:05:03');
        −62; 3
```

- `PERIOD_ADD(P,N)` Ajoute `N` mois à la date `P` (au format `YYMM` ou `YYYYMM`). Retourne le résultat au format `YYYYMM`. Il faut bien noter que la date `P` n'est pas au format `date`.

```
mysql62; select PERIOD_ADD(9801,2);
        −62; 199803
```

- `PERIOD_DIFF(P1,P2)` Retourne le nombre de mois entre deux dates `P1` et `P2`. `P1` et `P2` doivent être au format . Il faut bien noter que les dates `P1` et `P2` ne sont pas au format `date`.

```
mysql62; select PERIOD_DIFF(9802,199703);
        −62; 11
```

- `DATE_ADD(date,INTERVAL expression type)`
- `DATE_SUB(date,INTERVAL expression type)`
- `ADDDATE(date,INTERVAL expression type)`
- `SUBDATE(date,INTERVAL expression type)` Ces fonctions effectuent des opérations arithmétiques sur les dates. Elles ont été inserées dans ***MySQL*** 3.22. `ADDDATE()` et `SUBDATE()` sont synonymes de `DATE_ADD()` et `DATE_SUB()`. `date` est de type `DATETIME` ou `DATE` , qui indique la date de début. `expression` est une expression qui donne une durée à ajouter ou à retrancher à la date de début. `expression` est une chaîne qui peut commencer par un signe moins (``−''), pour indiquer une durée négative. `type` est un mot clé qui indique comment l'expression doit être considéré. La table suivante établit la relation `type` et `expression` :

`type` ***value***	***Meaning***	***Expected*** `expr` ***format***
`SECOND`	Seconds	`SECONDS`
`MINUTE`	Minutes	`MINUTES`
`HOUR`	Hours	`HOURS`
`DAY`	Days	`DAYS`
`MONTH`	Months	`MONTHS`

YEAR	Years	YEARS
MINUTE_SECOND	Minutes and seconds	"MINUTES:SECONDS"
HOUR_MINUTE	Hours and minutes	"HOURS:MINUTES"
DAY_HOUR	Days and hours	"DAYS HOURS"
YEAR_MONTH	Years and months	"YEARS−MONTHS"
HOUR_SECOND	Hours, minutes,	"HOURS:MINUTES:SECONDS"
DAY_MINUTE	Days, hours, minutes	"DAYS HOURS:MINUTES"
DAY_SECOND	Days, hours, minutes, seconds	"DAYS HOURS:MINUTES:SECONDS"

MySQL allows any non−numeric delimiter in the `expr` format. The ones shown in the table are the suggested delimiters. If the `date` argument is a `DATE` value and your calculations involve only `YEAR`, `MONTH` and `DAY` parts (that is, no time parts), the result is a `DATE` value. Otherwise the result is a `DATETIME` value.

```
mysql62; SELECT DATE_ADD("1997−12−31 23:59:59",
                         INTERVAL 1 SECOND);
        −62; 1998−01−01 00:00:00
mysql62; SELECT DATE_ADD("1997−12−31 23:59:59",
                         INTERVAL 1 DAY);
        −62; 1998−01−01 23:59:59
mysql62; SELECT DATE_ADD("1997−12−31 23:59:59",
                         INTERVAL "1:1" MINUTE_SECOND);
        −62; 1998−01−01 00:01:00
mysql62; SELECT DATE_SUB("1998−01−01 00:00:00",
                         INTERVAL "1 1:1:1" DAY_SECOND);
        −62; 1997−12−30 22:58:59
mysql62; SELECT DATE_ADD("1998−01−01 00:00:00",
                         INTERVAL "−1 10" DAY_HOUR);
        −62; 1997−12−30 14:00:00
mysql62; SELECT DATE_SUB("1998−01−02", INTERVAL 31 DAY);
        −62; 1997−12−02
mysql62; SELECT EXTRACT(YEAR FROM "1999−07−02");
       −62; 1999
mysql62; SELECT EXTRACT(YEAR_MONTH FROM "1999−07−02 01:02:03");
       −62; 199907
mysql62; SELECT EXTRACT(DAY_MINUTE FROM "1999−07−02 01:02:03");
       −62; 20102
```

MySQL accepte l'utilisation de délimiteurs non−numériques dans le format de `expression` . Ceux présentés dans le tableaux ne sont que des suggestions. Si l'argument `date` est au format `DATE` et que les calculs font intervenir seulement `YEAR`, `MONTH` et `DAY` (c'est à dire, juste la date, par l'heure.), le résultat est de type `DATE` . Sinon, le résultat des de type `DATETIME` .

```
mysql62; select DATE_ADD("1997−12−31 23:59:59",
                         INTERVAL 1 SECOND);
        −62; 1998−01−01 00:00:00
mysql62; select DATE_ADD("1997−12−31 23:59:59",
                         INTERVAL 1 DAY);
        −62; 1998−01−01 23:59:59
mysql62; select DATE_ADD("1997−12−31 23:59:59",
                         INTERVAL "1:1" MINUTE_SECOND);
        −62; 1998−01−01 00:01:00
mysql62; select DATE_SUB("1998−01−01 00:00:00",
                         INTERVAL "1 1:1:1" DAY_SECOND);
        −62; 1997−12−30 22:58:59
mysql62; select DATE_ADD("1998−01−01 00:00:00",
                         INTERVAL "−1 10" DAY_HOUR);
        −62; 1997−12−30 14:00:00
mysql62; select DATE_SUB("1998−01−02", INTERVAL 31 DAY);
        −62; 1997−12−02
```

Si l'argument de durée est trop court par rapport au mot clé spécifié, ***MySQL*** suppose que les parties de

gauche sont mises à zéro. Par exemple, avec le mot clé DAY_SECOND , s'attend à trouver le format "JOURS HEURES:MINUTES:SECONDES" . Si l'argument de durée est "1:10", supposera que les jours et heures sont à 0, et que seules, les minutes et secondes sont fournies. En un sens, ***MySQL*** ignore le type spécifié, et utilise à la place MINUTE_SECOND. Si les dates sont incorrectes, le résultat est NULL. Par ailleurs, lors de l'utilisation des options MONTH, YEAR_MONTH ou YEAR, et que dans le mois résultant, la date du jours n'existe pas, elle est automatiquement ramenée à la plus grande valeur qu'elle peut prendre dans ce mois.

```
mysql62; select DATE_ADD('1998−01−30', Interval 1 month);
        −62; 1998−02−28
```

Note from the preceding example that the word INTERVAL and the type keyword are not case sensitive.

- TO_DAYS(date) Retourne l'index du jour par rapport au 1er janvier 0.

```
mysql62; select TO_DAYS(950501);
        −62; 728779
mysql62; select TO_DAYS('1997−10−07');
        −62; 729669
```

TO_DAYS() n'est pas prévu pour utiliser des dates précédents l'avènement du calendrier Grégorien. (1582)..

- FROM_DAYS(N) Etant donné un numéro de jour par rapport au 1er janvier 0, retourne une valeur de type DATE.

```
mysql62; select FROM_DAYS(729669);
        −62; '1997−10−07'
```

FROM_DAYS()n'est pas prévu pour utiliser des dates précédents l'avènement du calendrier Grégorien. (1582).

- DATE_FORMAT(date,format) Formate la date date en fonction de la chaîne format. Les formats suivants peuvent être utilisé dans format :

%M	Month name (January..December)
%W	Weekday name (Sunday..Saturday)
%D	Day of the month with english suffix (1st, 2nd, 3rd, etc.)
%Y	Year, numeric, 4 digits
%y	Year, numeric, 2 digits
%a	Abbreviated weekday name (Sun..Sat)
%d	Day of the month, numeric (00..31)
%e	Day of the month, numeric (0..31)
%m	Month, numeric (01..12)
%c	Month, numeric (1..12)
%b	Abbreviated month name (Jan..Dec)
%j	Day of year (001..366)
%H	Hour (00..23)
%k	Hour (0..23)
%h	Hour (01..12)
%I	Hour (01..12)
%l	Hour (1..12)
%i	Minutes, numeric (00..59)
%r	Time, 12−hour (hh:mm:ss [AP]M)
%T	Time, 24−hour (hh:mm:ss)
%S	Seconds (00..59)

%s	Seconds (00..59)
%p	AM or PM
%w	Day of the week (0=Sunday..6=Saturday)
%U	Week (0..52), where Sunday is the first day of the week
%u	Week (0..52), where Monday is the first day of the week
%%	A literal `%'.

Tous les autres caractères sont recopiés, sans interpretation

```
mysql62; select DATE_FORMAT('1997−10−04 22:23:00', '%W %M %Y');
        −62; 'Saturday October 1997'
mysql62; select DATE_FORMAT('1997−10−04 22:23:00', '%H:%i:%s');
        −62; '22:23:00'
mysql62; select DATE_FORMAT('1997−10−04 22:23:00',
                            '%D %y %a %d %m %b %j');
        −62; '4th 97 Sat 04 10 Oct 277'
mysql62; select DATE_FORMAT('1997−10−04 22:23:00',
                            '%H %k %I %r %T %S %w');
        −62; '22 22 10 10:23:00 PM 22:23:00 00 6'
```

Depuis ***MySQL*** 3.23, le caractère % est obligatoire devant le caractère de format. Dans les versions antérieures de ***MySQL***, % il était optionnel.

- `TIME_FORMAT(time,format)` Utilisation identique à `DATE_FORMAT()`, mais seulement pour les heures (heures, minutes secondes). Les autres arguments conduisent à un résultat a `NULL` ou `0`.
- `CURDATE()`
- `CURRENT_DATE` Retourne la date du jour, au format `'YYYY−MM−DD'` ou `YYYYMMDD` , suivant que la fonction est utilisée en contexte chaîne ou numérique

```
mysql62; select CURDATE();
        −62; '1997−12−15'
mysql62; select CURDATE() + 0;
        −62; 19971215
```

- `CURTIME()`
- `CURRENT_TIME` Retourne l'heure du jour, au format `'HH:MM:SS'` or `HHMMSS`, suivant que la fonction est utilisée en contexte chaîne ou numérique

```
mysql62; select CURTIME();
        −62; '23:50:26'
mysql62; select CURTIME() + 0;
        −62; 235026
```

- `NOW()`
- `SYSDATE()`
- `CURRENT_TIMESTAMP` Retourne la date et l'heure du jour, au format `'YYYY−MM−DD HH:MM:SS'` or `YYYYMMDDHHMMSS`, suivant que la fonction est utilisée en contexte chaîne ou numérique

```
mysql62; select NOW();
        −62; '1997−12−15 23:50:26'
mysql62; select NOW() + 0;
        −62; 19971215235026
```

- `UNIX_TIMESTAMP()`
- `UNIX_TIMESTAMP(date)` Utilisé sans argument, retourne un timestamp Unix (le nombre de secondes depuis `'1970−01−01 00:00:00'` GMT) . Utilisé avec un argument de type `date`, il renvoie le timestamp Unix correspondant à cette date. `DATE` peut être aux formats chaîne `DATE` chaîne, `DATETIME` chaîne, `TIMESTAMP`, ou un nombre au format `YYMMDD` ou `YYYYMMDD`.

```
mysql62; select UNIX_TIMESTAMP();
        −62; 882226357
mysql62; select UNIX_TIMESTAMP('1997−10−04 22:23:00');
        −62; 875996580
```

Quand un `UNIX_TIMESTAMP` est affecté à une colonne de type `TIMESTAMP`, l'affectation sera directe, avec aucune conversion implicite.

- FROM_UNIXTIME(unix_timestamp) Retourne la représentation de l'argument `unix_timestamp` au format `'YYYY−MM−DD HH:MM:SS'` or `YYYYMMDDHHMMSS` suivant que la fonction est utilisée en contexte chaîne ou numérique

```
mysql62; select FROM_UNIXTIME(875996580);
        −62; '1997−10−04 22:23:00'
mysql62; select FROM_UNIXTIME(875996580) + 0;
        −62; 19971004222300
```

- `FROM_UNIXTIME(unix_timestamp,format)` Retourne la représentation de l'argument `unix_timestamp` au format `format`, suivant que la fonction est utilisée en contexte chaîne ou numérique. Le format est spécifié comme pour la fonction the `DATE_FORMAT()`.

```
mysql62; select FROM_UNIXTIME(UNIX_TIMESTAMP(),
                              '%Y %D %M %h:%i:%s %x');
        −62; '1997 23rd December 03:43:30 x'
```

- `SEC_TO_TIME(seconds)` Converti l'argument `seconds` en heures, minutes et secondes, au format `'HH:MM:SS'` or `HHMMSS`, suivant que la fonction est utilisée en contexte chaîne ou numérique

```
mysql62; select SEC_TO_TIME(2378);
        −62; '00:39:38'
mysql62; select SEC_TO_TIME(2378) + 0;
        −62; 3938
```

- `TIME_TO_SEC(time)` Retourne l'argument `time`, converti en secondes.

```
mysql62; select TIME_TO_SEC('22:23:00');
        −62; 80580
mysql62; select TIME_TO_SEC('00:39:38');
        −62; 2378
```

7.3.12 Fonctions diverses

- `DATABASE()` Retourne le nom de la base de données courante

```
mysql62; select DATABASE();
        −62; 'test'
```

S'il n'y a pas de base de données courante, `DATABASE()` retourne une chaîne vide.

- USER()
- `SYSTEM_USER()`
- `SESSION_USER()` Retourne le nom de l'utilisateur ***MySQL*** courant.

```
mysql62; select USER();
        −62; 'davida@localhost'
```

Avec ***MySQL*** 3.22.11 ou plus récent, le nom de l'utilisateur courant contient le nom du de l'hôte client. On peut alors en extraire le nom du User.

```
mysql62; select left(USER(),instr(USER(),"@")−1);
        −62; 'davida'
```

- `PASSWORD(str)` Calcule un mot de passe à partir du texte de `str`. C'est une fonction d'encryption utilisée par ***MySQL*** pour stocker les mots de passes dans la colonne `Password`, de la table des droits des utilisateurs.

```
mysql62; select PASSWORD('badpwd');
        −62; '7f84554057dd964b'
```

Le cryptage de `PASSWORD()` n'est pas reversible. `PASSWORD()` effectue un cryptage différent de celui d'Unix. Il ne faut pas croire que si le mot de passe de ***MySQL*** et d'Unix sont les mêmes, alors les deux valeurs cryptées seront les mêmes. Voir `ENCRYPT()`.

- `ENCRYPT(str[,salt])` Crypte la chaîne `str` en utilisant le cryptage d'Unix(`crypt()`)l. Le grain de sel est une chaîne avec deux caractères.

```
mysql62; select ENCRYPT("hello");
        −62; 'VxuFAJXVARROc'
```

Si la commande `crypt()` n'est pas disponible sur votre système, `ENCRYPT()`retournera `NULL`. `ENCRYPT()` n'utilise que les 8 premiers caractères de la chaîne, sur certains systèmes. Cela dépendra du comportement de l'appel système : `crypt()`.

- ENCODE(str,pass_str) Crypte la chaîne `str` en utilisant de mot de passe `pass_str` . Pour décrypter le résultat, il faut utiliser `DECODE()`. Le résultat est une chaîne binaire. Pour l'enregistrer, il faut l'affecter à une colonne de type `BLOB` .
- `DECODE(crypt_str,pass_str)` Décrypte la chaîne cryptée `crypt_str`, en utilisant le mot de passe `pass_str` . `crypt_str` doit être une chaîne retournée par `ENCODE()`.
- `MD5(string)` Calcule la somme de vérification MD5 de ma chaîne. La valeur retournée est un nombre hexadécimal de 32 bits, utilisé comme clé.

```
mysql62; select MD5("testing")
        −62; 'ae2b1fca515949e5d54fb22b8ed95575'
```

Cette fonction est issue de "RSA Data Security, Inc. MD5 Message−Digest Algorithm".

- `LAST_INSERT_ID([expression ])` Retourne la dernière valeur générée automatiquement lors de la mise à jour d'une colonne de type `AUTO_INCREMENT` Voir Section `mysql_insert_id.`

```
mysql62; select LAST_INSERT_ID();
        −62; 195
```

Le dernier ID généré est conservé par le serveur, sur la base d'un par connexion. Il ne sera pas changé par un autre client. Il ne sera même pas changé par la mise à jour d'une colonne de type `AUTO_INCREMENT`. Si `expression` est donné comme argument à `LAST_INSERT_ID()` dans une clause , alors la valeur de l'argument est retournée, en tant que `LAST_INSERT_ID()`. Cela peut être utile pour simuler des séquences. Par exemple :

- Création d'une table :

```
mysql62; create table sequence (id int not null);
mysql62; insert into sequence values (0);
```

Alors la table peut être utilisée pour générer une séquence de nombre comme ceci :

```
mysql62; update sequence set id=LAST_INSERT_ID(id+1);
```

Il est possible de générer des séquences sans appeler `LAST_INSERT_ID()`, mais l'interêt de cette fonction et que l'ID est conservé par le serveur en tant que dernière valeur générée. Il est ainsi possible d'obtenir un nouvel ID comme lors de la lecture de n'importe quelle colonne de type `AUTO_INCREMENT`. Par exemple, `LAST_INSERT_ID()` (sans argument) retournera un nouvel ID . La méthode C API `mysql_insert_id()` peut aussi être utilisée pour obtenir une telle valeur.

- FORMAT(X,D) Met le nombre `X` au format `'#,###,###.##'` avec `D` décimales. Si `D` vaut `0`, le résultat n'aura ni décimales, ni virgule.

```
mysql62; select FORMAT(12332.1234, 2);
        −62; '12,332.12'
mysql62; select FORMAT(12332.1,4);
        −62; '12,332.1000'
mysql62; select FORMAT(12332.2,0);
        −62; '12,332'
```

- `VERSION()` Retourne la version du serveur ***MySQL*** .

```
mysql62; select VERSION();
        −62; '3.22.19b−log'
```

- `GET_LOCK(str,timeout)` Essaie d'obtenir le verrous nommé `str` , avec un timeout de `timeout` secondes. Retourne 1 si le verrous a pu être obtenu, 0 en cas d'échec, ou si une erreur est survenue (comme par exemple, plus de mémoire disponible, ou le thread a été tué par l'administrateur). Un verrou est libéré avec la fonction `RELEASE_LOCK()`, avec l'exécution de la fonction `GET_LOCK()` ou la mort du thread courant. Cette fonction peut être utilisée pour implémenter des verrous d'applications, ou simuler des verrous sur les enregistrements.

```
mysql62; select GET_LOCK("lock1",10);
        −62; 1
mysql62; select GET_LOCK("lock2",10);
```

```
        −62; 1
mysql62; select RELEASE_LOCK("lock2");
        −62; 1
mysql62; select RELEASE_LOCK("lock1");
        −62; NULL
```

Il faut noter que le deuxième appel à `RELEASE_LOCK()` retourne `NULL` car le verrou `"lock1"` a été automatiquement libéré par le deuxième appel à `GET_LOCK()`.

- RELEASE_LOCK(str) Libère le verrous nommé `str`, obtenu par l'appel de `GET_LOCK()` . Retourne 1 si le verrous a bien été libéré, et 0 si il n'était pas mis par ce thread (dans ce cas, il reste verrouillé), et si ce verrous n'existe pas. Le verrous n'existe pas tant qu'il n'a pas été créer par `GET_LOCK()` ou si il a déjà été libéré.
- BENCHMARK(count,expression) La fonction de `BENCHMARK` exécute l'expression `expression count` fois. Cela permet de mesurer le temps que ***MySQL*** met à exécuter l'expression. Le résultat est toujours `0`. Le temps mis pour l'exécution de la commande est disponible sur l'affichage du client ***MySQL***.

```
mysql62; select BENCHMARK(1000000,encode("hello","goodbye"));
+----------------------------------------------+
| BENCHMARK(1000000,encode("bonjour","ca va")) |
+----------------------------------------------+
|                                            0 |
+----------------------------------------------+
1 row in set (4.74 sec)
```

Le temps affiché sur le client est le temps entre le début et la fin de l'exécution, et non pas le temps de travail du processeur. Il peut être nécessaire d'éxécuter plusieurs fois la commande, pour prendre en compte la charge de la machine.

7.3.13 Fonctions à utiliser dans les clauses `GROUP BY`

L'utilisation d'une fonction de regroupement dans une commande qui ne contient pas la clause `GROUP BY` est équivalent à regrouper toutes les lignes.

- `COUNT(expression )` Retourne le nombre de lignes non−`NULL` obtenue par une commande `SELECT`.

```
mysql62; select student.student_name,COUNT(*)
           from student,course
           where student.student_id=course.student_id
           GROUP BY student_name;
```

`COUNT(*)` est optimisé pour compter très rapidement les lignes obtenues par un `SELECT` sur une seule table, sans qu'aucune autre colonne ne soit demandée, et sans clause `WHERE` . Par exemple :

```
mysql62; select COUNT(*) from student;
```

- `COUNT(DISTINCT expression ,[expression ...])` Retourne le nombre de ligne distinctes.

```
mysql62; select COUNT(DISTINCT results) from student;
```

Avec ***MySQL*** il est possible d'obtenir le nombre de combinaison d'expressions distinctes en donnant une liste d'expression. E, ANSI SQL, il aurait fallu concaténer les expressions avec `CODE(DISTINCT ..)`.

- `AVG(expression )` Retourne la moyenne des valeurs de `expression` .

```
mysql62; select student_name, AVG(test_score)
           from student
           GROUP BY student_name;
```

- `MIN(expression )`
- `MAX(expression )` Retourne le minimum ou le maximum de `expression` . `MIN()` et `MAX()` peuvent travailler avec des chaînes. Dans ce cas, il retourne la chaîne minimum maximum .

```
mysql62; select student_name, MIN(test_score), MAX(test_score)
           from student
           GROUP BY student_name;
```

- `SUM(expression )` Retourne la somme de `expression` . Si l'ensemble n'a aucune ligne, le résultat est `NULL`!
- `STD(expression )`
- `STDDEV(expression )` Retourne la déviation standard de `expression` . C'est une extension à la norme ANSI SQL. La fonction `STDDEV()` est fourni pour assurer la comptabilité avec les bases Oracle.
- `BIT_OR(expression )` Retourne le OU logique bit−à−bit, effectué sur `expression` . Ce calcul est fait sur 64 bits (précision de `BIGINT`).
- `BIT_AND(expression )` Retourne le ET logique bit−à−bit, effectué sur `expression` . Ce calcul est fait sur 64 bits (précision de `BIGINT`).

MySQL permet une utilisation étendue de `GROUP BY`. Il est possible de faire des calculs sur des colonnes dans le `SELECT` même si elles n'apparaissent pas dans le `GROUP BY` . Cela est possible pour n'importe quelle valeur de ce groupe. Cela permet de gagner en performance en évitant de faire des regroupements ou des tris sur des valeurs inutiles. Par exemple, il n'y a pas besoin de faire un regroupement avec `customer.name` dans la requête suivante :

```
mysql62; select order.custid,customer.name,max(payments)
       from order,customer
       where order.custid = customer.custid
       GROUP BY order.custid;
```

La norme ANSI SQL impose d'ajouter `customer.name` dans la clause `GROUP BY` . Avec ***MySQL***, c'est redondant.

Il ne faut pas utiliser cette particularité si les noms de colonnes ne sont pas unique dans le groupe courant.

Dans certains cas, il est possible d'utiliser `MIN()` et `MAX()` pour obtenir la valeur d'une colonne spécifique, même si elle n'est pas unique. Par exemple, cette requête retourne la valeur de `column`, de la ligne qui contient la colonne `sort` la plus courte.

```
substr(MIN(concat(sort,space(6−longueur(sort)),column),7,longueur(column)))
```

Il faut noter que avec ***MySQL*** 3.22 (ou avant), ou en suivant la norme ANSI SQL, il ne faut pas utiliser d'expressions dans les clauses `GROUP BY` ou `ORDER BY`. Il faut alors contourner la difficulté en utilisant un alias.

```
mysql62; SELECT id,FLOOR(value/100) AS val FROM Nom_table
           GROUP BY id,val ORDER BY val;
```

Avec ***MySQL*** 3.23, on peut écrire :

```
mysql62; SELECT id,FLOOR(value/100) FROM Nom_table ORDER BY RAND();
```

7.4 CREATE DATABASE

```
CREATE DATABASE nom_base_de_donnees
```

CREATE DATABASE Nom_bdd `CREATE DATABASE` crée une base de donnée avec le nom Nom_bdd. Les règles qui régissent les nom de base de données sont détaillées dans la section 7.1.5 Noms de base de données, table, index, column et alias. ***MySQL*** retourne une erreur si la base de données existe déjà. Avec ***MySQL***, les bases sont représentées sous la forme de dossiers qui contiendront des fichiers, qui représenteront les tables. Etant donné qu'à la création d'une base, cette dernière est vide, `CREATE`

DATABASE se contente de créer un dossier dans le dossier données de ***MySQL***. Il est aussi possible de créer des bases avec la méthode `mysqladmin.` 12.1 Présentation des différents programmes MySQL.

7.5 DROP DATABASE

```
DROP DATABASE [IF EXISTS] Nom_bdd
```

DROP DATABASE efface toutes les tables de la base , et efface aussi la base, i.e. le dossier. Il faut être TRES prudent avec cette commande.

DROP DATABASE retourne le nombre de fichiers qui ont été effacés. Normalement, ce nombre correspond à trois fois le nombre de table, étant donné que chaque table a trois fichier, un fichier ``.ISD',' un fichier ``.ISM'' et un fichier ``.frm''.

MySQL 3.22 ou plus récent autorise l'utilisation du mot clé `IF NOT EXISTS`, qui ne retourne pas d'erreur si la base à effacer n'existe pas.

Il est aussi possible d'effacer des bases avec la méthode mysqladmin. 12.1 Présentation des différents programmes MySQL.

7.6 CREATE TABLE

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] Nom_table (create_definition,...)
[options_de_table] [commande_de_selection]

create_definition:
  Nom_col type [NOT NULL | NULL] [DEFAULT valeur_par_defaut] [AUTO_INCREMENT]
              [PRIMARY KEY] [reference_definition]
  ou    PRIMARY KEY (index_Nom_col,...)
  ou    KEY [Nom_index] KEY(index_Nom_col,...)
  ou    INDEX [Nom_index] (index_Nom_col,...)
  ou    UNIQUE [INDEX] [Nom_index] (index_Nom_col,...)
  ou    [CONSTRAINT symbole] FOREIGN KEY Nom_index(index_Nom_col,...)
              [reference_definition]
  ou    CHECK (expression )

type:
        TINYINT[(longueur)] [UNSIGNED] [ZEROFILL]
  ou    SMALLINT[(longueur)] [UNSIGNED] [ZEROFILL]
  ou    MEDIUMINT[(longueur)] [UNSIGNED] [ZEROFILL]
  ou    INT[(longueur)] [UNSIGNED] [ZEROFILL]
  ou    INTEGER[(longueur)] [UNSIGNED] [ZEROFILL]
  ou    BIGINT[(longueur)] [UNSIGNED] [ZEROFILL]
  ou    REAL[(longueur,décimales)] [UNSIGNED] [ZEROFILL]
  ou    DOUBLE[(longueur,décimales)] [UNSIGNED] [ZEROFILL]
  ou    FLOAT[(longueur,décimales)] [UNSIGNED] [ZEROFILL]
  ou    DECIMAL(longueur,décimales) [UNSIGNED] [ZEROFILL]
  ou    NUMERIC(longueur,décimales) [UNSIGNED] [ZEROFILL]
  ou    CHAR(longueur) [BINARY]
  ou    VARCHAR(longueur) [BINARY]
  ou    DATE
  ou    TIME
  ou    TIMESTAMP
  ou    DATETIME
  ou    TINYBLOB
```

```
  ou    BLOB
  ou    MEDIUMBLOB
  ou    LONGBLOB
  ou    TINYTEXT
  ou    TEXT
  ou    MEDIUMTEXT
  ou    LONGTEXT
  ou    ENUM(value1,value2,value3,...)
  ou    SET(value1,value2,value3,...)

index_Nom_col:
        Nom_col [(longueur)]

reference_definition:
        REFERENCES Nom_table [(index_Nom_col,...)]
                   [MATCH FULL | MATCH PARTIAL]
                   [ON DELETE reference_option]
                   [ON UPDATE reference_option]

reference_option:
        RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT

options_de_table:
type = [ISAM | MYISAM | HEAP]

ouauto_increment = #
ouavg_row_longueur = #
ouchecksum = [0 | 1]
oucomment = "string"
oulignes_max= #
oulignes_min = #
oupack_keys = [0 | 1]
oupassword= "string"
commande_de_selection:
[ | IGNORE | REPLACE] SELECT ...  (Un select valide)
```

`CREATE TABLE` crée une table, de nom Nom_table, dans la base de donnés courante. Les règles qui régissent les nom de table sont détaillées dans 7.1.5 Noms de base de données, table, index, column et alias. ***MySQL*** retourne une erreur si il n'y a pas de base de données courante, ou la table existe déjà.

Avec ***MySQL*** 3.22 ou plus récent, on peut se référer à la table en utilisant la structure `Nom_bdd.Nom_table`. Ceci fonctionne, qu'il y ait une base de données courante ou pas.

MySQL 3.23 autorise l'utilisation du mot clé `TEMPORARY` lors de la création de table. Une table temporaire sera automatiquement effacée lorsque la connexion sera terminée. Cela permet à deux connexions différents d'utiliser le même nom de table temporaire, sans conflit l'un avec l'autre, ou avec une table existante (la table permanente est cachée jusqu'à ce que la table temporaire soit effacée).

MySQL 3.22 ou plus récent autorise l'utilisation du mot clé `IF NOT EXISTS`, qui ne retourne pas d'erreur si la table à créer existe déjà.

Chaque table est representée par un ou plusieurs fichiers dans le dossier de la bas e de données. Dans le cas ou la table est de type ISAM, il y aura :

Pour plus d'information sur les propriétés des différents type de colonne, 7.2 Types de colonnes.

- Si ni `NULL` ni `NOT NULL` n'a été précisé, alors la colonne est considéré comme de type `NULL`.
- Une colonne de type entier peut avoir l'attribut `AUTO_INCREMENT`. A chaque insertion de la valeur ou de 0 dans une colonne de type

AUTO_INCREMENT, la valeur de la colonne est mise à value+1, avec value qui est la plus grande valeur dans la colonne de la table courante. AUTO_INCREMENT commence à 1. Si la ligne contenant la plus grande valeur de la colonne est effacée, cette valeur sera réutilisée. Si toutes les lignes de la table sont effacée, AUTO_INCREMENT recommence à 1. Il ne peux y avoir qu'une seule colonne de type AUTO_INCREMENT par table, et cette colonne doit être indexée. Par souci de compatibilité avec ODBC, il est possible d'accéder à la dernière ligne insérée avec la requête suivante :

```
SELECT * FROM Nom_table WHERE auto_col IS NULL
```

- Les valeurs NULL sont gérées de manière différentes pour les colonnes de type TIMESTAMP. Il n'est pas possible de stocker la valeur NULL dans une colonne de type TIMESTAMP, alors cette affectation conduit à affecter la date et l'heure courante à la place. A cause de ce comportement, les attributs NULL et NOT NULL ne s'appliquent pas, et sont ignorées. D'un autre coté, afin de rendre ***MySQL*** plus simple à utiliser, le serveur autorise l'affectation de NULL aux colonnes de type TIMESTAMP (ce qui est vrai), même si les colonnes de type TIMESTAMP ne contiendront jamais réellement la valeur NULL. Cette information est accessible en utilisant la commande DESCRIBE Nom_table qui détaille les colonnes de la table. Il est bon de rappeler que la valeur 0 est une valeur valide de TIMESTAMP.
- Si aucune valeur par défaut n'est précisé avec l'attribut DEFAULT, ***MySQL*** en assignera automatiquement une. Si la colonne peut prendre la valeur NULL, alors ***MySQL*** utilisera cette valeur par défaut. Sinon, son comportement dépendra du type de colonne :
  - Pour les types numériques autres que ceux ayant l'attribut AUTO_INCREMENT , la valeur par défaut est 0 . Pour les colonnes ayant l'attribut AUTO_INCREMENT, la valeur par défaut est la prochaine valeur de la séquence.
  - Pour les types date et heures, excepté TIMESTAMP, la valeur par défaut est le ``zéro'' . Pour la première colonne de type TIMESTAMP, la valeur par défaut est la date et l'heure courante. 7.2.6 Types date et heure.
  - Pour les types chaînes, autre que ENUM, la valeur par défaut est la chaîne vide. Pour le type ENUM, la valeur par défaut est la première valeur de l'énumération.
- KEY est un synonyme pour INDEX.
- Avec ***MySQL***, l'attribut UNIQUE force la valeur à toujours prendre une valeur distincte. Une erreur surviendra lors de l'insertion d'une ligne qui doublera une ligne déjà existante.
- Avec ***MySQL,*** l'attribut PRIMARY KEY est identique à l'attribut KEY qui porterait le nom de PRIMARY Une table ne peut avoir qu'une seule colonne avec l'attribut table PRIMARY KEY. Si aucune colonne n'a de PRIMARY KEY et qu'une application requiert la colonne de PRIMARY KEY, ***MySQL*** retournera la première colonne ayant l'attribut UNIQUE.
- Une PRIMARY KEY peut être un index multi colonne. Mais il n'est pas possible de créer un index multi colonne qui aurait l'attribut PRIMARY KEY. Pour cela, il faut utiliser la fonction PRIMARY KEY(index_Nom_col, ...) , sous peine de ne voir que la première colonne porter l'attribut PRIMARY KEY.
- Si aucun nom n'est affecté à un index, un nom sera automatiquement généré et assigné. Ce nom sera constitué du nom de la première colonne, avec un suffixe optionnel,(_2, _3, ...), pour l rendre unique. La liste des noms d'index est accessible avec la requête : SHOW INDEX FROM Nom_table.
- Seule les tables de type MyISAM acceptent les index sur les colonnes qui contiennent la valeur NULL. Dans les autres cas, il faut absolument déclarer la colonne de type NOT NULL pour ne pas générer une erreur.
- Avec la syntaxe Nom_col(longueur), il est possible de spécifier un index qui utilise seulement une partie de la colonne de type CHAR or VARCHAR. Cela rend l'index nettement plus petit et plus efficace. Indexes.
- Seules les tables de type MyISAM acceptent d'indexer les colonnes de type BLOB et TEXT . mettre un index sur une telle colonne impose de préciser la longueur de la colonne dans l'index.

```
CREATE TABLE test (blob_col BLOB, index(blob_col(10)));
```

- Lors de l'utilisation des clauses ORDER BY ou GROUP BY avec de colonnes de type TEXT ou BLOB, seuls les max_sort_longueur premiers octets sont pris en compte. BLOB.
- Les attributs FOREIGN KEY, CHECK et REFERENCES ne font en réalité rien de spécial. Ils sont uniquement fournis pour assurer la compatibilité et la portabilité de programme SQL et de les faire tourner avec ***MySQL***.
- Les colonnes de type prennent un bit de plus, arrondi à l'octet le plus proche.
- La taille maximale d'un enregistrement peut être obtenue avec la requête suivante :

```
row longueur = 1
             + (sum of column longueurs)
             + (number of NULL columns + 7)/8
             + (number of variable−longueur columns)
```

- Les options options_de_table et SELECT ne sont disponibles qu'à partir de ***MySQL*** 3.23. Les différents types de tables sont :

Les options de table sont utilisées pour optimiser la gestion de la table. Dans la plus part des cas, ces options peuvent être ignorées. Ces options fonctionnent pour toutes les types de table, sauf indication contraire.

Lors de l'utilisation d'une tape de type MyISAM table, ***MySQL*** utilise le produit max* avg_row_longueur pour connaître la taille maximale de la table. Par défaut, la taille maximale de la table est 4Go (4 Giga octets) (ou 2Go, si le système ne les supporte pas.

- Si un CREATE STATEMENT est spécifié après SELECT, ***MySQL*** va créer de nouveaux champs pour chacun des éléments dans le SELECT . Par exemple

```
mysql62; CREATE TABLE test (a int not null auto_increment, primary key (a), key(b))
TYPE=HEAP SELECT b,c from test2;
```

Cette requête va créer une table de type HEAP table avec 3 colonnes. Cette table sera automatiquement effacée si une erreur survient lors de la copie des données dans ma table.

7.6.1 Modifications automatiques de type de colonne

Dans certains cas, ***MySQL*** spontanément les spécifications d'une colonne lors d'une commande de CREATE TABLE (Cela peut aussi intervenir lors d'une commande with ALTER TABLE.)

- Les colonnes de type VARCHAR d'une longueur de moins de 4 caractères sont changées en CHAR.
- Si l'une des colonne de la table a une longueur variable, alors la ligne complète a aussi une longueur variable. Par conséquent, si la table contient une colonne de longueur variable(VARCHAR, TEXT or BLOB), toutes les colonnes de type CHAR et de longueur supérieur à 3 caractères sont changées en colonne de type VARCHAR .Cela ne change en rien l'utilisation de ces colonnes, cela modifie simplement la façon dont les colonnes sont stockées. ***MySQL*** effectue cette conversion car cela économise de la place, et rend les opérations sur la table plus rapides.
- La taille d'affichage des colonnes de type TIMESTAMP doit être pair et compris dans l'intervalle de 2 à 14. En dehors de l'intervalle 2 à 14, la taille de l'affichage est ramenée à 14. En cas de nombre impair, le nombre est arrondi au nombre pair directement supérieur.
- Il est impossible d'enregistrer une valeur NULL dans une colonne de type TIMESTAMP, car une telle affectation revient à assigner la date et l'heure courante à la colonne. De ce fait, les attributs NULL et NOT NULL ne s'appliquent pas aux colonnes de type TIMESTAMP, et sont ignorés. Lors d'une requête DESCRIBE Nom_table ***MySQL*** répondra toujours que la colonne accepte les valeurs de type NULL.
- ***MySQL*** effectue la correspondance entre certains types utilisé par d'autres bases de données SQL vers des types ***MySQL***. 7.2.11 Utiliser des types de colonnes d'autres bases de données.

Pour savoir si ***MySQL*** a modifié spontanément le type d'une colonne, il faut utiliser la requête DESCRIBE Nom_table après la création ou la modification de la table.

Certaines autres modifications de type peuvent intervenir lors de la compression de table avec la commande pack_isam. 10.17 Quel sont les différents formats de lignes? Quand utiliser VARCHAR/CHAR?.

7.7 ALTER TABLE

```
ALTER [IGNORE] TABLE Nom_table alter_spec [, alter_spec ...]
alter_specification:
        ADD [COLUMN] create_definition [FIRST | AFTER column_name ]
  ou    ADD INDEX [Nom_index] (index_Nom_col,...)
  ou    ADD PRIMARY KEY (index_Nom_col,...)
  ou    ADD UNIQUE [Nom_index] (index_Nom_col,...)
  ou    ALTER [COLUMN] Nom_col {SET DEFAULT literal | DROP DEFAULT}
  ou    CHANGE [COLUMN] old_Nom_col create_definition
  ou    MODIFY [COLUMN] create_definition
  ou    DROP [COLUMN] Nom_col
  ou    DROP PRIMARY KEY
  ou    DROP INDEX key_name
  ou    RENAME [AS] new_Nom_table
  ou    table_option
```

ALTER TABLE modifie la structure d'une table existante. Par exemple, il est possible d'ajouter ou d'effacer des colonnes, de créer et détruire des index, de changer le type d'une colonne ou de la renommer, voire même de renommer la table elle−même. Il est aussi possible de changer le commentaire de la table et son type. Voir CREATE TABLE.

Si, lors d'une modification de type de colonne, la commande DESCRIBE Nom_table indique que la

colonne n'a pas changé de type, il est possible que ***MySQL*** ait ignoré la modification pour une des raisons décrites dans 7.6.1 Modifications automatiques de type de colonne. Par exemple, si il existe plusieurs colonne de type de longueur variable, rien de sert de tenter de forcer une colonne du type VARCHAR à CHAR.

ALTER TABLE fonctionne en effectuant une copie temporaire de la table originale. La modification est effectuée sur la copie, puis l'originale est remplacée par la copie modifiée. Toutes les mises à jours sont automatiquement appelé, et sont faites dans un mode sans erreur. Pendant la modification, la table originale est toujours accessible en lecture par les autres clients. Les mises à jour et les écritures sont reportées jusqu'à la fin de la modification.

- Pour pouvoir utiliser la commande `ALTER TABLE`, il faut avoir les droits pour ***select***, ***insert***, ***delete***, ***update***, ***create*** et ***drop*** sur la table.
- `IGNORE` est une extension ***MySQL*** de la norme ANSI SQL92. Il permet de contrôler la façon avec laquelle réagit si il trouve des doublons dans une colonne de clés uniques. Si n'est pas précisé, la copie est annulée à la première erreur, et la modification est annulée. Si est précisé, alors la première occurrence d'une clés en double sera utilisée, les autres occurrences étant ignorées et effacées.
- Il est possible d'utiliser plusieurs clauses `ADD`, `ALTER`, `DROP` et `CHANGE` dans une commande `ALTER TABLE`. C'est une extension de ***MySQL*** à la norme ANSI SQL92 : cette dernière ne permet qu'une seule de ces clauses à chaque commande.
- `CHANGE Nom_col`, `DROP Nom_col` et `DROP INDEX` sont des extensions de ***MySQL*** à la norme ANSI SQL92
- `CHANGE Nom_col`, `DROP Nom_col` et `DROP INDEX` sont des extensions de ***MySQL*** à la norme ANSI SQL92 .
- `MODIFY` est une extension Oracle à `ALTER TABLE`.
- L'option `COLUMN` est simplement de la décoration et peut être ignoré.
- La commande " `ALTER TABLE Nom_table RENAME AS new_name` " sans aucune autre option, permet de renommer la table. ***MySQL*** va simplement renommer les fichiers correspondant à la table `Nom_table`. Il n'y a pas de création de table temporaire.
- La clause `create_definition` utilise la même syntaxe que les clauses `ADD` et `CHANGE`, ainsi que `CREATE TABLE.` Cette inclus le nom de la colonne, et pas seulement le type de colonne. Voir `CREATE TABLE`.
- La commande `CHANGE old_Nom_col create_definition` permet de changer le nom d'une colonne. Pour cela, il faut spécifier l'ancien et le nouveau nom de la colonne, ainsi que le type courant de la colonne. Par exemple, pour renommer une colonne de type `INTEGER` de 'a' en 'b', la commande suivante est valable :

```
mysql62; ALTER TABLE t1 CHANGE a b INTEGER;
```

Pour changer le type de la colonne mais pas son nom, la syntaxe de CHANGE requiert deux noms de colonnes, même si ils sont identiques. Par exemple :

```
mysql62; ALTER TABLE t1 CHANGE b b BIGINT NOT NULL;
```

Cependant, à partir de ***MySQL*** 3.22.16a, il est possible d'utiliser la clause `MODIFY` pour changer le type de la colonne sans le renommer:

```
mysql62; ALTER TABLE t1 MODIFY b BIGINT NOT NULL;
```

- Lors de l'utilisation de `CHANGE` ou `MODIFY` pour réduire la taille d'une colonne qui possède un index (par exemple, un index sur les 10 premiers caractères d'une colonne de type `VARCHAR`), il est impossible de rendre la colonne plus petite que ne le requiert l'index.
- En changeant le type d'une colonne avec `CHANGE` ou `MODIFY`, ***MySQL*** essaie de convertir au mieux les informations d'un type à l'autre.
- A partir de ***MySQL*** 3.22, il est possible d'utiliser `FIRST` ou `ADD ... AFTER Nom_col` pour ajouter une colonne à une position donnée, dans une table. Par défaut, l'ajout de fait après la dernière colonne.
- `ALTER COLUMN` spécifie une nouvelle valeur par défaut, ou bien efface l'ancienne valeur. Si la valeur par défaut de la colonne est effacée, et que la colonne peut être `NULL`, alors la nouvelle valeur par défaut est `NULL`.Si la colonne ne peut pas être `NULL`, alors ***MySQL*** assigne une valeur par défaut arbitraire, comme décrit dans le paragraphe `CREATE TABLE`.
- `DROP INDEX` efface un index. C'est une extension de ***MySQL*** à la norme ANSI SQL92 .
- Lorsqu'une colonne est effacée d'une table, la colonne est aussi effacée de toutes les index qui l'utilise. Si toutes les colonnes qui composent un index sont effacées, l'index disparaît aussi.
- `DROP PRIMARY KEY` efface la colonne qui porte l'attribut `PRIMARY KEY` . Si une telle colonne n'existe pas, la première colonne de type `UNIQUE` est effacée à la place. (***MySQL*** utilise la première colonne `UNIQUE` comme `PRIMARY KEY` si aucune `PRIMARY KEY` n'est spécifiée.)
- Avec l'API C `mysql_info()`, il est possible de savoir combien de ligne ont été copiée, et (quand l'option `IGNORE` était mise), le nombre de lignes qui furent effacées, à cause de la duplication de clés.
- Les attributs `FOREIGN KEY`, `CHECK` et `REFERENCES` ne font en réalité rien de spécial. Ils sont uniquement fournis pour assurer la compatibilité et la portabilité de programme SQL et de les faire tourner avec ***MySQL***.

Voici quelques exemples d'utilisation de la commande ALTER TABLE. On supposera que la table `t1`, créée ci−dessous, existe :

```
mysql62; CREATE TABLE t1 (a INTEGER,b CHAR(10));
```

Renomme la table de `t1` et `t2`:

```
mysql62; ALTER TABLE t1 RENAME t2;
```

Change le type de la colonne `a` de `INTEGER` en `TINYINT NOT NULL` (mais ne change pas le nom), et change le type de la colonne `b` de `CHAR(10)` en `CHAR(20)` et renomme cette colonne en `c`.

```
mysql62; ALTER TABLE t2 MODIFY a TINYINT NOT NULL, CHANGE b c CHAR(20);
```

Ajoute une nouvelle colonne `d`, de type `TIMESTAMP`.

```
mysql62; ALTER TABLE t2 ADD d TIMESTAMP;
```

Ajoute un index sur la colonne `d`, et fait de la colonne a une `PRIMARY KEY`

```
mysql62; ALTER TABLE t2 ADD INDEX (d), ADD PRIMARY KEY (a);
```

Efface la colonne `c`:

```
mysql62; ALTER TABLE t2 DROP COLUMN c;
```

Ajoute une nouvelle colonne de type `AUTO_INCREMENT integer` , nommée `c`:

```
mysql62; ALTER TABLE t2 ADD c INT UNSIGNED NOT NULL AUTO_INCREMENT,
          ADD INDEX (c);
```

Il faut remarquer que la colonne `c` a été indexée, car les colonnes de type `AUTO_INCREMENT` doivent être indexée, ce qui implique `c` doit avoir l'attribut `NOT NULL`, puisque les colonnes indexées ne peuvent pas avoir de valeur à `NULL`.

7.8 OPTIMIZE TABLE

```
OPTIMIZE TABLE Nom_table
```

`OPTIMZE TABLE` est à utiliser après avoir effacer de grandes parties d'une table, pour après des modifications majeures sur une table à longueur de ligne variable. Les index des lignes effacées sont conservés, et l'opération `INSERT` les réutilise, et écrase mes anciennes lignes. La commande `OPTIMIZE TABLE` permet de récupérer l'espace inutilisé des tables.

`OPTIMIZE TABLE` fonctionne en effectuant une copie temporaire de la table originale. La modification est effectuée sur la copie, puis l'originale est remplacée par la copie modifiée. Toutes les mises à jours sont automatiquement appelé, et sont faites dans un mode sans erreur. Pendant la modification, la table originale est toujours accessible en lecture par les autres clients. Les mises à jour et les écritures sont reportées jusqu'à la fin de la modification.

7.9 DROP TABLE

```
DROP TABLE [IF EXISTS] Nom_table [, Nom_table,...]
```

DROP TABLE efface une ou plusieurs tables. Toutes les données et le fichier de définition sont effacés. Cette commande est irréversible

MySQL 3.22 ou plus récent autorise l'utilisation du mot clé IF EXISTS, qui ne retourne pas d'erreur si la table à effacer n'existe pas.

7.10 DELETE

```
DELETE [LOW_PRIORITY] FROM Nom_table
    [WHERE where_definition] [LIMIT rows]
```

DELETE efface les lignes de la table Nom_table qui satisfont les conditions données par where_definition, et retourne le nombre de ligne effacées.

Une commande DELETE sans clause WHERE s'applique à toutes les lignes, c'est à dire que la table est vidée de toutes ses données (mais elle n'est pas effacée). ***MySQL*** recrée alors la table, mais vide, ce qui est nettement plus rapide que d'effacer les lignes unes à unes. Dans ce cas particulier, ***MySQL*** retourne 0 (. (***MySQL*** ne peut pas connaître le nombre de ligne de la table sans l'avoir ouverte, car la recréation est faite sans accéder à la table), return the number of rows that were actually deleted, since the recreate is done without opening the data files). Tant que le fichier ``Nom_table.frm' est valide, il est possible de recréer la table, même si le fichier de données est corrompu).'

Pour obtenir le nombre de ligne effacées lors de l'effacement de la table, il est toujours possible de sacrifier la vitesse d'exécution, et d'utiliser la commande suivante :

```
mysql62; DELETE FROM Nom_table WHERE 162;0;
```

Cette requête est particulièrement LENTE, car l'effacement se fait ligne à ligne, à cause de la clause WHERE

L'option LOW_PRIORITY permet de reporter l'effacement jusqu'au moment ou il ne reste plus personne qui lise la table.

Les ligne effacées sont conservées dans une liste, et les insertions ultérieurs réutiliseront cette place. Pour forcer la récupération de cette place, il faut utiliser la commande OPTIMIZE TABLE ou bien l'utilitaire isamchk pour réorganiser les tables. OPTIMIZE TABLE est plus simple, mais isamchk plus rapide. OPTIMIZE TABLE.

L'option LIMIT, spécifique à ***MySQL,*** permet d'indiquer au serveur le nombre maximum de ligne à effacer. Cela permet d'éviter qu'une commande DELETE ne prenne trop de temps. Il suffit alors de répéter la commande jusqu'à ce qu'elle ait effacé moins de ligne que LIMIT.

7.11 SELECT

```
SELECT [STRAIGHT_JOIN] [SQL_SMALL_RESULT] [DISTINCT | DISTINCTROW | ALL]
    select_expression,...
    [INTO OUTFILE 'Nom_fichier' export_options]
    [FROM table_references
        [WHERE where_definition]
        [GROUP BY Nom_col,...]
        [HAVING where_definition]
        [ORDER BY {unsigned_integer | Nom_col | formula} [ASC | DESC] ,...]
        [LIMIT [offset,] rows]
        [PROCEDURE procedure_name] ]
```

SELECT est utilisé pour obtenir des lignes à partir d'une ou plusieurs tables.
select_expression indique les colonnes à lire. SELECT peut aussi être utilisé pour exécuter des calculs sans rapport avec aucune table. Par exemple :

```
mysql62; SELECT 1 + 1;
         −62; 2
```

Toutes les options doivent impérativement être dans l'ordre indiqué ci dessus. Par exemple, une clause HAVING doit être après GROUP BY et avant la clause ORDER BY .

- Une clause SELECT peut utiliser des alias, introduit par le mot clé AS. L'alias est utilisé comme un nom de colonne, et peut être repris dans une clause ORDER BY ou HAVING. Par exemple:

```
mysql62; select concat(last_name,', ',first_name) AS full_name
    from maTable ORDER BY full_name;
```

- La clause FROM table_references indique les noms des tables qu'il faut interroger. Si il y a plusieurs tables, il vaut mieux utiliser la clause join. JOIN.
- Il est possible d'appeler une colonne sous les formes Nom_col, Nom_table.Nom_col or Nom_bdd.Nom_table.Nom_col. Il n'y a pas besoin de préciser les préfixes Nom_table ou Nom_bdd.Nom_table lors d'un SELECT à moins que les noms soient ambiguës. 7.1.5 Noms de base de données, table, index, column et alias, pour des exemples sur les ambiguïtés qui peuvent apparaître.
- Une référence sur une table peut être aliasée avec Nom_table [AS] Nom_alias.

```
mysql62; select t1.name, t2.salary from employee AS t1, info AS t2
           where t1.name = t2.name;
mysql62; select t1.name, t2.salary from employee t1, info t2
           where t1.name = t2.name;
```

- Les colonnes demandées en sortie peuvent être utilisées dans les clauses ORDER BY et GROUP BY en utilisant leur nom, leurs alias ou leur positions. Les colonnes sont positionnées à partir de 1.

```
mysql62; select college, region, seed from tournament
           ORDER BY region, seed;
mysql62; select college, region AS r, seed AS s from tournament
           ORDER BY r, s;
mysql62; select college, region, seed from tournament
           ORDER BY 2, 3;
```

Pour trier dans l'ordre descendant, il faut ajouter le mot clé DESC après le nom de la colonne, dans la clause ORDER BY . Par défaut, l'ordre ascendant est utilisé, et peut être explicitement demandé en utilisant le mot clé ASC

- La clause HAVING peut faire référence à n'importe quelle colonne ou alias présent dans select_expression. Cette clause est évaluée en dernier, juste avant que les lignes soient envoyées au client, sans aucune optimisation. Il ne faut pas utiliser HAVING là WHERE où est plus efficace. Par exemple, il ne faut pas écrire :

```
mysql62; select Nom_col from Nom_table HAVING Nom_col 62; 0;
```

A la place, il vaut mieux écrire :

```
mysql62; select Nom_col from Nom_table WHERE Nom_col 62; 0;
```

A partir de ***MySQL*** 3.22.5, on peut écrire des requêtes telles que:

```
mysql62; select user,max(salary) from users
          group by user HAVING max(salary)62;10;
```

Dans les versions anciennes de ***MySQL*** , il était possible d'écrire :

```
mysql62; select user,max(salary) AS sum from users
          group by user HAVING sum62;10;
```

- L'option `STRAIGHT_JOIN` force l'optimiseur à regrouper les tables dans l'ordre dans lequel elles sont spécifiées dans la `FROM` . Cela permet d'accélérer le traitement d'une requête. `EXPLAIN.`
- L'option `SQL_SMALL_RESULT` peut être utilisé avec les clauses `GROUP BY` ou `DISTINCT` pour indiquer à l'optimiseur que le résultat sera de petite taille. Dans ce cas, ***MySQL*** va utiliser des tables temporaires d'accès rapide, pour enregistrer les tables de résultat, plutot que de faire des tries. `SQL_SMALL_RESULT` est une extension ***MySQL*** de ANSI SQL92.
- L'option `LIMIT` peut être utilisée pour restreindre le nombre de lign retournée `SELECT` . `LIMIT` a un ou deux arguments numériques.. Le premier indique l'index de la ligne de début, et le deuxième indique le nombre de ligne à retourner. L'indexe de la ligne initiale est 0.

```
mysql62; select * from table LIMIT 5,10;  # retourne les lignes 6−15
```

Si un seul argument est fourni à LIMIT, il indique le nombre de ligne à retourner.

```
mysql62; select * from table LIMIT 5;     # retourne les 5 premiere lignes
```

`LIMIT n` et `LIMIT 0,n` sont équivalents.

- La forme `SELECT ... INTO OUTFILE 'Nom_fichier'` de la syntaxe de `SELECT` écrit les lignes selectionnées dans un fichier. Les fichiers créés sont écrit sur le serveur, et ne doit pas exister au moment de l'écriture (cela évite d'écraser le fichier /etc/passwd, par exemple). Il faut avoir les droits d'écriture. `SELECT ... INTO OUTFILE` est le contraire de la commande `LOAD DATA INFILE`; qui importe les lignes à partir d'un fichier. L'expression `export_options` est constituée des même champs `FIELDS` et `LINES` que dans la commande `LOAD DATA INFILE` . section `LOAD DATA`. Dans le fichier texte résultant de la commande `SELECT ... INTO OUTFILE` , seuls, les caractères suivants sont précédé du caractère d'échappement, précisé avec `ESCAPED BY`:
  - ♦ Le caractère `ESCAPED BY`
  - ♦ Le premier caractère dans la clause `FIELDS TERMINATED BY`
  - ♦ Le premier caractère dans la clause `LINES TERMINATED BY`

De plus, le caractère `ASCII 0` est converti en `ESCAPED BY` suivi de 0 (`ASCII 48`). La raison de l'ajout du caractère d'échappement après les caractères `FIELDS TERMINATED BY`, `ESCAPED BY` ou `LINES TERMINATED BY` ,est que cela permet la relecture du fichier. Le caractère `ASCII 0` est échappé pour le rendre plus lisible par les éditeurs. Et comme le fichier résultant n'a pas à se conformer à la norme SQL, ce sont les seuls caractères à être échappés.

7.12 JOIN

MySQL utilise les syntaxes suivantes pour les commandes de `JOIN`:

```
table_reference, table_reference
table_reference [CROSS] JOIN table_reference
table_reference INNER JOIN table_reference
table_reference STRAIGHT_JOIN table_reference
table_reference LEFT [OUTER] JOIN table_reference ON conditional_expression
table_reference LEFT [OUTER] JOIN table_reference USING (column_list)
table_reference NATURAL LEFT [OUTER] JOIN table_reference
{ oj table_reference LEFT OUTER JOIN table_reference ON conditional_expression  }
```

La dernière syntaxe `LEFT OUTER JOIN` n'existe que pour assurer la compatibilité avec ODBC.

- Les références sur les tables peuvent être des alias.

```
mysql62; select t1.name, t2.salary from employee AS t1, info AS t2
           where t1.name = t2.name;
```

- `INNER JOIN` et `,` (comma) sont équivalents. Les deux effectuent un regroupement des tables utilisées. Normalement, il faut préciser comment les tables sont reliées avec la condition `WHERE` .
- La condition `ON` est identique à `WHERE.`
- Si il n'y a pas de lignes qui correspondent à la table de gauche, dans la clause `LEFT JOIN`, une ligne dont toutes les colonnes sont mises à `NULL` est générée. Ceci permet de rechercher les lignes d'une table qui n'ont pas de contrepartie dans une autre table.

```
mysql62; select table1.* from table1
           LEFT JOIN table2 ON table1.id=table2.id
           where table2.id is NULL;
```

Cet exemple recherche toutes les lignes dans la `table1` avec une colonne `id` qui n'est pas présent dans la table `table2` (i.e., toutes les lignes de la table `table1` qui n'ont pas de ligne correspondantes dans la table `table2`). Cela implique que a été déclaré `table1.id`, bien entendu !

- La clause `USING column_list` spécifie une liste de nom de colonne qui doivent exister dans toutes les tables. Une utilisation telle que

```
A LEFT JOIN B USING (C1,C2,C3,...)
```

Correspond à l'utilisation d'une clause `ON` comme ceci :

```
A.C1=B.C1 AND A.C2=B.C2 AND A.C3=B.C3,...
```

- La clause `NATURAL LEFT JOIN` de deux tables est équivalent à utilisation de la clause `LEFT JOIN` avec `USING`, en précisant les noms de toutes les colonnes qui existent dans les deux tables.
- `STRAIGHT_JOIN` est identique à `JOIN`, à l'exception du fait que la table de gauche est lue avant la table de droite. Cela est pratique pour les (rares) cas où l'optimiseur de regroupement utilise les tables dans le mauvais ordre .

Quelques exemples

```
mysql62; select * from table1,table2 where table1.id=table2.id;
mysql62; select * from table1 LEFT JOIN table2 ON table1.id=table2.id;
mysql62; select * from table1 LEFT JOIN table2 USING (id);
mysql62; select * from table1 LEFT JOIN table2 ON table1.id=table2.id
           LEFT JOIN table3 ON table2.id=table3.id;
```

`LEFT JOIN` optimization.

7.13 `INSERT`

```
    INSERT [LOW_PRIORITY | DELAYED] [IGNORE]
        [INTO] Nom_table [(Nom_col,...)]
        VALUES (expression,...),(...),...
ou  INSERT [LOW_PRIORITY | DELAYED] [IGNORE]
        [INTO] Nom_table [(Nom_col,...)]
        SELECT ...
ou  INSERT [LOW_PRIORITY | DELAYED] [IGNORE]
        [INTO] Nom_table
        SET Nom_col=expression, Nom_col=expression, ...
```

`INSERT` insère une nouvelle ligne dans une table existante.. La forme de `INSERT ... VALUES` est basée sur des colonnes explicitement précisée. forme de `INSERT ... SELECT` insère des données depuis une autre table. La forme de `INSERT ... VALUES` avec plusieurs valeurs est acceptée ***MySQL*** à partir de la version 3.22.5 . La syntaxe `Nom_col=expression` est accepté par ***MySQL*** à partir de la version 3.22.10.

`Nom_table` est le nom de la table dans laquelle les lignes vont être insérée. La liste de nom de colonne ou la clause `SET` indique quelles colonnes vont être assignées.

- Si aucun nom de colonne n'est précisé dans la commande, `INSERT ... VALUES` or `INSERT ... SELECT`, alors des valeurs doivent être fournies pour toutes les colonnes dans la liste de `VALUES()`. Pour connaître l'ordre des colonnes, il faut utiliser la commande `DESCRIBE Nom_table`.
- Toute colonne qui n'a pas de valeur explicitement fournie est mise à sa valeur par défaut. Par exemple, il est possible de fournir une liste de colonne en omettant certains noms : les valeurs de ces colonnes seront les valeurs par défaut. Pour avoir les détails sur les valeurs par défaut, `CREATE TABLE`.
- Une `expression` peut faire référence à n'importe quelle colonne déjà nommée dans la liste des colonnes. Par exemple :

```
mysql62; INSERT INTO Nom_table (col1,col2) VALUES(15,col1*2);
```

Mais pas ceci :

```
mysql62; INSERT INTO Nom_table (col1,col2) VALUES(col2*2,15);
```

- L'option `LOW_PRIORITY` permet de différer une exécution jusqu'à ce qu'il n'y ait plus de client qui lisent la table.
- L'option permet, lors de l'insertion dans une table qui a une colonne de type `PRIMARY` or `UNIQUE` , de ne pas renvoyer d'erreur si une insertion essaie de doubler une clé déjà existante. Si cette option n'est pas précisée, l'insertion est annulée à partir de la ligne erronée. Le nombre de ligne correctement insérées est accessible avec `mysql_info()` .
- Si ***MySQL*** a été configuré avec l'option `DONT_USE_DEFAULT_FIELDS` , une commande `INSERT` devra avoir la liste complète des colonnes qui requière une valeur non−`NULL`.
- Les conditions suivantes s'appliquent aux commandes de type `INSERT INTO ... SELECT`
  - La requête ne peut pas contenir de clause `ORDER BY`
  - La table cible de l'insertion ne peut pas apparaître dans la clause `FROM` du `SELECT`, car la norme ANSI SQL l'interdit (en effet, le `SELECT` pourrait trouver des lignes qui viennent juste d'être insérée. Utiliser des sous−selections serait encore pire).
  - `Les colonnes de type AUTO_INCREMENT` fonctionnent de la même façon.

Lors de l'utilisation de `INSERT ... SELECT` ou `INSERT ... VALUES` avec des listes de plusieurs lignes, la fonction `mysql_info()` permet d'obtenir des informations sur la requête. Le format de la réponse est comme suite :

```
Records: 100 Duplicates: 0 Warnings: 0
```

`Duplicates` indique le nombre de lignes qui n'a pas pus être insérées car elles tentaient de doubler une clé primaire.. `Warnings` indique le nombre d'insertions qui ont générer une erreur lors de l'insertion. `Warnings` surviennent lorsqu'il y a une tentative:

- Insertion d'une valeur `NULL` dans une colonne déclarée `NOT NULL`.
- Assignation d'une valeur numérique qui est hors de l'intervalle de validité de la colonne. Cette valeur est ramenée à la valeur valide la plus proche.
- Assignation d'une valeur telle que `'10.34 a'`. Dans ce cas, les résidus inutiles sont éliminés. Si la valeur ne peut pas être interprétée, la valeur `0` est assignée à la place.
- Insertion d'une chaîne dans une colonne de type `CHAR`, `VARCHAR`, `TEXT` ou `BLOB` qui excède la longueur maximale. La valeur est alors tronquée à la taille maximale de la colonne.
- Insertion d'une valeur invalide dans une date ou heure. La valeur est alors fixée au ``zéro'' du type de la colonne.

L'option `DELAYED` pour la commande `INSERT` est une caractéristique ***MySQL*** qui est très utile lorsque les clients ne peuvent pas attendre la fin de l'insertion. C'est utilisé habituellement par ***MySQL*** pour remplir des historiques, et que périodiquement, une longue commande `SELECT` est effectuée. `DELAYED` a été introduit à partir de ***MySQL*** 3.22.15. C'est une extension ***MySQL*** à la norme ANSI SQL92.

Un autre avantage majeur de l'utilisation de la commande `INSERT DELAYED` est que les insertions sont regroupées et traitées en même temps. C'est une manière plus rapide que d'effectuer autant d'insertions unitaires.

Il faut noter que les insertions en attente sont gardées en mémoire vive, jusqu'à ce qu'elles soient insérées dans la table. Cela signifie que si le processus `mysqld` est brutalement interrompu (`kill −9`) ou si `mysqld` se termine inopinément, les lignes ne seront pas écrites sur le disque, et ainsi perdues !

Les événements suivants surviennent lors de l'utilisation de l'option DELAYED des commandes INSERT ou REPLACE. Dans la description qui suit, le ``thread'' est le thread qui a recu la commande INSERT DELAYED et ``handler'' est le thread qui va gérer les INSERT DELAYED pour une table particulière.

- Quand un thread exécute la commande DELAYED pour une table, le thread handler est crée pour gérer toutes les commandes DELAYED pour la table, si un tel thread n'existe pas.
- Le thread vérifie si le handler a bien posé un verrou DELAYED, et si non, il lui dit de le faire. Le verrou DELAYED peut être obtenu même si d'autres threads ont déjà obtenu un verrou de READ ou WRITE sur la table. Cependant, le handler va attendre que tous les verrous ALTER TABLE ou FLUSH TABLES soient libérés, afin de s'assurer que la table est bien à jour.
- Le thread exécute l'insertion mais, au lieu d'écrire la ligne dans la table, il transmet une copie de la ligne finale au handler, qui l'ajoute dans la queue d'attente. Toutes les erreurs d'insertions sont notées par le thread, et rapportées au client.
- Le client ne reçoit pas le nombre de doublons, ou la valeur d' AUTO_INCREMENT qu'il pourrait attendre, car ces informations ne sont connues qu'après l'insertion proprement dite. De la même façon, mysql_info()risque de retourner des informations incohérentes.
- L'historique de mise à jour est modifié par le handler quand la ligne est effectivement insérée dans la table. Si il y a plusieurs lignes insérées simultanément, l'historique de mise à jour est modifié lors de l'insertion de la première ligne.
- Après chaque bloc de delayed_insert_limit lignes écrites, le handler vérifie si il n'y a pas de commande SELECT en attente. Si c'est le cas, il s'arrête, et l'exécute avant de continuer.
- Quand le handler n'a plus de ligne dans la queue, la table est déverrouillée. Si aucune autre commandes INSERT DELAYED n'est reçue dans un délai de delayed_insert_timeout secondes, le handler se termine.
- Si il reste plus de delayed_queue_size lignes en attente d'insertion, le thread attend jusqu'à ce qu'il y ait de la place dans la queue. Ce permet de contrôler la quantité de mémoire utilisé pour les queues d'attentes.
- Le handler apparaîtra dans la liste des processus ***MySQL*** avec le mot delayed_insert dans la colonne de Command . Il sera automatiquement effacé lors d'une commande FLUSH TABLES ou KILL thread_id. Cependant, il enregistrera les lignes dans la table avant de quitter. Durant cette période, le handler n'acceptera plus aucune nouvelle commande INSERT d'un autre thread. Si une nouvelle commande an INSERT DELAYED est éxécutée, un nouveau handler sera créé.
- Il faut bien noter que les commandes INSERT DELAYED ont une priorité supérieure aux commandes INSERT, si un handler est déjà en fonctionnement. Toutes les autres commandes doivent attendre que la queue d'attente du handler soit vide (même par un kill).
- Les statuts suivants fournissent des informations sur l'état des commandes INSERT DELAYED :

Ces variables sont accessibles avec la commande SHOW STATUS ou, en ligne, avec mysqladmin extended−status.

7.14 REPLACE

```
    REPLACE [LOW_PRIORITY | DELAYED]
        [INTO] Nom_table [(Nom_col,...)]
        VALUES (expression,...)
ou  REPLACE [LOW_PRIORITY | DELAYED]
        [INTO] Nom_table [(Nom_col,...)]
        SELECT ...
ou  REPLACE [LOW_PRIORITY | DELAYED]
        [INTO] Nom_table
        SET Nom_col=expression, Nom_col=expression,...
```

REPLACE fonctionne exactement comme INSERT, mais si une ligne d'une table a une colonne qui porte l'attribut unique , REPLACE pourra remplacer cette ligne par une nouvelle ligne, tout en gardant la même valeur dans la colonne de type unique. Pour plus de détails, voir INSERT.

7.15 LOAD DATA INFILE

```
LOAD DATA [LOCAL] INFILE 'file_name.txt' [REPLACE | IGNORE]
    INTO TABLE nom_table
    [FIELDS
        [TERMINATED BY '\t']
        [OPTIONALLY] ENCLOSED BY '']
        [ESCAPED BY '\\' ]]
    [LINES TERMINATED BY '\n']
```

```
[IGNORE number LINES]
[(nom_colonne,...)]
```

La commande `LOAD DATA INFILE` lit des lignes à partir d'un fichier, et le transforme en table, à très grande vitesse. Si l'option `LOCAL` est précisée, le fichier est lu depuis le client (cette fonction est disponible à partir la version 3.22.6 de ***MySQL***.)

Pour des raisons de sécurité, lors de la lecture de fichier situé sur le serveur, les fichiers doivent être disponibles dans le dossier de ***MySQL***, ou bien lisible par tous. De plus, pour utiliser `LOAD DATA INFILE` sur des fichiers serveurs, il faut avoir les droits ***fichiers***.

Utiliser l'option `LOCAL` est un peu plus lente que l'option par défaut, car le contenu des fichiers doit du client vers le serveur. D'un autre coté, il n'y plus de problème de droits d'accès.

L'utilitaire `mysqlimport` assure aussi l'importation de fichier. Il le fait en envoyant une requête `LOAD DATA INFILE` au serveur. L'option `–local` force `mysqlimport` à lire le fichier depuis l'hôte client. Si le client et le serveur supporte le protocole compressé, l'option `––compress` donnera de meilleures performances sur des réseaux chargés.

Pour retrouver les fichiers sur le serveur, le serveur utilise les règles suivantes :

- Si un chemin absolu est fourni, le serveur utilise le chemin tel quel.
- Si un chemin relatif est fourni, avec un ou plusieurs composant, le serveur recherche le fichier dans le dossier données de la base de données.
- Si un nom de fichier est fourni, le serveur va le rechercher dans le dossier de la base de données courante.

Il faut noter que ces règles signifie que le fichier ``` ``./monFichier.txt'' ``` sera lu depuis le dossier données du serveur, tandis que ``` ``myfile.txt'' ``` sera lu depuis le dossier de la base de données courante. Il faut aussi noter qu'avec la commande ci−dessous, le fichier est lu depuis le dossier de la base `db1`, et non pas `db2` :

```
mysql62; USE db1;
mysql62; LOAD DATA INFILE "./data.txt" INTO TABLE db2.my_table;
```

Les options `REPLACE` et `IGNORE` règlent la gestion index redondants, dans les colonnes de type unique. Avec `REPLACE`, la nouvelle ligne remplacera l'ancienne, avec la même valeur d'index. Avec `IGNORE`, la nouvelle ligne sera ignorée. Si rien n'est précisé, une erreur surviendra lors de la tentative d'insertion du doublon, et le reste du fichier sera ignoré.

Lors du chargement de lignes à partir d'un fichier local et avec l'option `LOCAL`, le serveur n'a aucun moyen d'interrompre la transmission du fichier durant l'opération, alors le comportement par défaut est `IGNORE`.

`DATA INFILE` est la commande complémentaire de `SELECT ... INTO OUTFILE`. SELECT. Pour écrire des lignes depuis une base de données vers un fichier, il faut utiliser `SELECT ... INTO OUTFILE`. Pour lire des lignes depuis un fichier vers une base, il faut utiliser `LOAD DATA INFILE`. La syntaxe des clauses `FIELDS` et `LINES` est la même pour les deux commandes. Ces deux clauses sont optionnelles, mais `FIELDS` doit impérativement précéder `LINES` si les deux sont présents.

Si la clause est `FIELDS` présente, alors chacune des sous clauses (`TERMINATED BY`, `[OPTIONALLY] ENCLOSED BY` t `ESCAPED BY`) sont optionnelles, mais il faut au moins en spécifier une.

Si la clause `FIELDS` n'est pas spécifiée, les valeurs par défaut sont les suivantes :

```
FIELDS TERMINATED BY '\t' ENCLOSED BY '' ESCAPED BY '\\'
```

Si la clause `LINES` n'est pas spécifiée, les valeurs par défaut sont les suivantes :

```
LINES TERMINATED BY '\n'
```

En bref, les options par défaut de `LOAD DATA INFILE` fonctionnent comme suit :

- Recherche de la limite de ligne
- Segmenter la ligne en champs, gr ce aux tabulations
- Ne pas s'attendre à ce que les champs soient entourés de guillemets
- Interpréter les occurrences de tabulation, nouvelle ligne, ou backslash ``\'' précédés par ``\'' comme des caractères à part entières

Dans le sens inverse, les options par défaut de `SELECT ... INTO OUTFILE` fonctionnent comme suit :

- Ecrire des tabulations entre les champs
- Ne pas entourer les champs avec des guillemets
- Ajouter un backslash ``\'' avant toutes les occurrences de tabulation, nouvelle ligne, ou backslash ``\'.''
- Ecrire une nouvelle ligne à chaque fin de ligne.

Il faut noter que pour écrire write `FIELDS ESCAPED BY '\\'`, il faut écrire 2 backslash pour en avoir un de lu.

L'option `IGNORE number LINES` permet d'ignorer les premières lignes, qui contiendrait un entête, par exemple.

```
mysql62; LOAD DATA INFILE "/tmp/Nom_fichier" into table test IGNORE 1 LINES;
```

Pour pouvoir écrire un fichier avec `SELECT ... INTO OUTFILE`, puis le relire `LOAD DATA INFILE` ultérieurement avec, il est impératif que les options de lecture et d'écriture soient les mêmes. Sinon, l'interprétation du fichier à la relecture sera erronée. Par exemple, un fichier est écrit avec `SELECT ... INTO OUTFILE` avec des virgules comme délimiteur de champs :

```
mysql62; SELECT * FROM table1 INTO OUTFILE 'data.txt'
           FIELDS TERMINATED BY ','
           FROM ...
```

Pour relire ce fichier, la bonne commande est :

```
mysql62; LOAD DATA INFILE 'data.txt' INTO TABLE table2
           FIELDS TERMINATED BY ',';
```

Si, au contraire, le fichier est relu avec la commande ci−dessous, la relecture sera erronée, car les délimiteurs attendus sont des tabulations :

```
mysql62; LOAD DATA INFILE 'data.txt' INTO TABLE table2
           FIELDS TERMINATED BY '\t';
```

Il est probable que le fichier soit interprété comme un seul champs.

`LOAD DATA INFILE` peut aussi lire des fichiers issues d'autres sources. Par exemple, un fichier au format dBase a des champs séparés par des virgules, et insérés dans des doubles guillemets. Si les lignes dans le fichier sont terminées par des nouvelles lignes, la commande suivante permettra d'acquérir un fichier au format.

```
mysql62; LOAD DATA INFILE 'data.txt' INTO TABLE Nom_table
           FIELDS TERMINATED BY ',' ENCLOSED BY '"'
           LINES TERMINATED BY '\n';
```

Les options FIELDS ou LINES peuvent être des chaînes vides. Si ils ne sont pas vides, les options FIELDS [OPTIONALLY] ENCLOSED BY et FIELDS ESCAPED BY doivent être une chaîne d'un seul caractère. Les options FIELDS TERMINATED BY et LINES TERMINATED BY peuvent avoir un ou plusieurs caractères. Par exemple, si les lignes sont terminées par la paire retour−chariot/nouvelle−ligne, il est possible d'utiliser l'option LINES TERMINATED BY '\r\n'.

Contrôle les caractères qui entoure les champs. Lors de l'exportation(SELECT ... INTO OUTFILE), l'absence de l'option OPTIONALLY force tous les champs à être entouré par le caractère ENCLOSED BY. Par exemple, en utilisant la virgule comme délimiteur de champs :

```
"1","une chaîne","100.20"
"2"," une chaîne contenant une , virgule","102.20"
"3"," une chaîne contenant un \" guillemet","102.20"
"4"," une chaîne contenant un \", guillemet et une virgule","102.20"
```

L'option OPTIONALLY force l'utilisation du caractère ENCLOSED BY seulement pour les champs de type CHAR et VARCHAR.

```
1," une chaîne ",100.20
2," une chaîne contenant une , virgule ",102.20
3," une chaîne contenant un \" guillemet ",102.20
4," une chaîne contenant un \", guillemet et une virgule ",102.20
```

On peut noter que les occurrences du caractère ENCLOSED BY situés dans une chaîne sont toujours échappée, gr ce au caractère ENCLOSED BY. On peut aussi noter que si le caractère d'échappement est une chaîne vide, le fichier ne pourra pas être relu correctement par LOAD DATA INFILE. Par exemple, le fichier de sortie ci−dessus, va devenir le fichier de sortie ci−dessous, si le caractère d'échappement est vide. Lors de la relecture, un problème surviendra surement durant la deuxième ligne :

```
1," une chaîne ",100.20
2," une chaîne contenant une , virgule ",102.20
3," une chaîne contenant un " guillemet ",102.20
4," une chaîne contenant un \", guillemet et une virgule ",102.20
```

En entrée, le caractère est éliminé à la fin de chaque champs. (ceci est vrai, qu'il y ai l'option OPTIONALLY ou pas. OPTIONALLY n'a pas d'impact sur la procédure d'acquisition). Les occurrences du caractère ENCLOSED BY précédés du caractère d'échappement ESCAPED BY sont considérés comme une partie du champs. De plus, les caractères ENCLOSED BY doublés sont considérés comme une seule occurrence. Par exemple, le caractère ENCLOSED BY est '"' alors, les lignes suivantes deviennent :

```
"Le ""GRAND"" chef"  −62; Le "GRAND" chef
Le "GRAND" chef −62; Le "GRAND" chef
Le ""GRAND"" chef −62; Le ""GRAND"" chef
```

FIELDS ESCAPED BY contrôle l'écriture et la lecture des caractères spéciaux . Si l'option FIELDS ESCAPED BY n'est pas une chaîne vide, il sert de préfixe dans les cas suivants

- Pour le caractère FIELDS ESCAPED BY
- Pour le caractère FIELDS [OPTIONALLY] ENCLOSED BY
- Pour le premier caractère FIELDS TERMINATED BY et LINES TERMINATED BY
- Pour le caractère ASCII 0 (qui sera en fait écrit avec la séquence caractère d'échappement suivi de 0)

Si le caractère `FIELDS ESCAPED BY` est une chaîne vide, aucun caractère ne sera échappé. Ce n'est pas une très bonne idée, surtout si certains champs contiennent l'un des caractères de la liste ci−dessus.

En lecture, si le caractère `FIELDS ESCAPED BY` n'est pas une chaîne vide, les occurrences de ce caractère seront éliminées, et le caractère suivant sera lu littéralement, comme une partie du champs. Exceptions faites de ``0'' et ``N'' (i.eI., `\0` ou `\N` avec ``\' 'comme caractère d'échappement). Ces séquences seront interprétées comme ASCII `0` (le caractère nul) et `NULL`. Voir ci dessous pour les règles de sauvegarde `NULL` .

Pour plus d'informations à propos des séquences d'échappement, Literals.

Dans certains cas, les options `FIELDS` et `LINES` de interagissent :

- Si `LINES TERMINATED BY` est une chaîne vide, et `FIELDS TERMINATED BY` est une chaîne non vide, les lignes seront aussi terminées `FIELDS TERMINATED BY`.
- Si `FIELDS TERMINATED BY` et `FIELDS ENCLOSED BY` sont tous les deux des chaînes vides (`''`), un format à longueur fixe est utilisé. Avec le format à longueur fixe, il n'y a plus besoin de délimiteurs. A la place, les colonnes et les lignes sont écrites en utilisant la taille d'affichage des colonnes. Par exemple, si une colonne est déclarée de type `INT(7)`, les valeurs seront écrite en utilisant des colonne de 7 caractères. En lecture, les données sont obtenues en lisant 7 caractères. Les fichiers sans délimiteurs ont un impact sur la façon avec laquelle la valeur `NULL` est enregistrée.

La gestion de la valeur `NULL` dépends des options `FIELDS` et `LINES` utilisées :

- Par pour les valeurs par défaut de `FIELDS` and `LINES`, `NULL` est symbolisé par as `\N` en lecture et en écriture. (en supposant que le caractère `ESCAPED BY` est ``\'').
- Si `FIELDS ENCLOSED BY` n'est pas vide, un champs contenant littéralement la chaîne `NULL` est utilisé, en écriture et en lecture. (Ceci est différent du mot `'NULL',` entouré de caractères `FIELDS ENCLOSED BY` et lu comme la chaîne `'NULL'`).
- Si `FIELDS ESCAPED BY` est vide, `NULL` est directement écrit `NULL`.
- Avec le format à largueur de colonne fixée, (ce qui arrive lorsque `FIELDS TERMINATED BY` et `FIELDS ENCLOSED BY` sont tous les deux vides), `NULL` s'écrit comme une chaîne vide. Il faut noter que cela entraîne l'identité de la valeur `NULL` et des chaînes vides. Il ne sera alors pas possible d'en faire la différence.

Certains cas ne sont pas accepté par `LOAD DATA INFILE`:

- Lignes de taille fixe (`FIELDS TERMINATED BY` et `FIELDS ENCLOSED BY` tous les deux vides) et des colonnes de type `BLOB` ou `TEXT`.
- Si deux séparateurs sont identiques, `LOAD DATA INFILE` ne seront pas capable d'interpréter le fichier. Par exemple, le champs `FIELDS` va poser problème:

```
FIELDS TERMINATED BY '"' ENCLOSED BY '"'
```

- Si `FIELDS ESCAPED BY` est vide, un champs qui contient une occurrence de `FIELDS ENCLOSED BY` ou `LINES TERMINATED BY` suivi `FIELDS TERMINATED BY` va provoquer une erreur, et l'arret de la lecture du fichier. Ceci, car `LOAD DATA INFILE` ne peut pas determiner correctement la fin de la valeur, ou du champs.

Les exemples suivants charge toutes les colonnes de la tables `persondata`:

```
mysql62; LOAD DATA INFILE 'persondata.txt' INTO TABLE persondata;
```

Aucun champs n'est précisé, alors `LOAD DATA INFILE` s'attend à trouver une valeur pour chaque colonne. Les valeurs par défaut `FIELDS` et `LINES` sont supposés.

Pour ne charger qu'une partie des colonnes, on peut utiliser :

```
mysql62; LOAD DATA INFILE 'persondata.txt'
           INTO TABLE persondata (col1,col2,...);
```

Il faut aussi préciser la liste des champs dans l'ordre d'apparition de celles ci dans le fichier, surtout si elles apparaissent dans un autre ordre que celui de la table.

Si il manque des champs, les colonnes qui n'ont pas de valeurs seront mises à leur valeur par défaut. Les valeurs par défaut sont décrites dans la section CREATE TABLE.

La valeur d'un champs vide est interprété différemment, suivant le champs manquant :

- Pour les champs de type chaîne, la colonne prend la valeur de la chaîne vide.
- Pour les types numériques, la colonne prend la valeur 0.
- Pour les types date et heures, la colonne prend la valeur ``zéro'' adéquate. 7.2.6 Types date et heure.

Les colonnes de type TIMESTAMP prennent la valeur de l'heure et la date courante si une NULL leur est affectée. ou (pour la première colonne de type TIMESTAMP) si la colonne est omise de la liste de champs à lire.

Si une ligne à trop de champs, les champs supplémentaires sont ignorés, et le nombre d'alerte est augmenté.

LOAD DATA INFILE considère toutes les valeurs en entrées comme des chaînes, donc il n'est pas possible d'utiliser les formes numériques, notamment pour les types ENUM ou SET . . Toutes les énumérations doivent être spécifiée comme des chaînes

Lors de l'utilisation de LOAD DATA INFILE, la fonction mysql_info() permet d'obtenir des informations sur la requête. Le format de la réponse est comme suite :

```
Records: 100 Duplicates: 0 Warnings: 0
```

Duplicates indique le nombre de lignes qui n'a pas pus être insérées car elles tentaient de doubler une clé primaire.. Warnings indique le nombre d'insertions qui ont générer une erreur lors de l'insertion. Warnings surviennent lorsqu'il y a une tentative incorrecte d'insertion de ligne.

7.16 UPDATE

```
UPDATE [LOW_PRIORITY] nom_table SET nom_colonne1=expr1,nom_colonne2=expr2,...
    [WHERE where_definition] [LIMIT #]
```

UPDATE met à jour une ligne éxistante dans une table. La clause SET indique quelles colonnes modifier, et quelles valeurs mettre dans ces colonnes. La conditions WHERE permet de choisir quelles lignes sont à mettre à jour. Sinon, toutes les lignes sont mises à jour

L'option LOW_PRIORITY, permet de retarder l'exécution de la requête jusqu'au moment où il n'y a plus de client qui lisent la table

Lors de l'accès à une colonne de la table Nom_table dans une expression, UPDATE utilise la valeur courante de la colonne. Par exemple, la requête suivante ajoute 1 à la colonne age.

```
mysql62; UPDATE persondata SET age=age+1;
```

Les commandes UPDATE sont évaluées de gauche à droite. Par exemple, la requête suivante double la colonne age, puis l'incrémente d'une unité :

```
mysql62; UPDATE persondata SET age=age*2, age=age+1;
```

Affecter la valeur courante d'une colonne lors d'une commande UPDATE conduit ***MySQL*** à ignore cette mise

à jour.

La commande UPDATE retourne le nombre de ligne qui ont été effectivement modifiées.. A partir de ***MySQL*** 3.22, la fonction C API `mysql_info()`nombre de ligne qui ont été trouvées et effectivement modifiées, puis le nombre de warnings de la commande UPDATE.

7.17 USE

```
USE Nom_bdd
```

La commande `USE Nom_bdd` statement indique à ***MySQL*** le nom de la base de données par défaut pour les requête suivantes. Cette base restera courante jusqu'à la fin de la session, ou jusqu'à la prochaine utilisation de la commande USE:

```
mysql62; USE db1;
mysql62; SELECT count(*) FROM maTable;      # selects from db1.maTable
mysql62; USE db2;
mysql62; SELECT count(*) FROM maTable;      # selects from db2.maTable
```

L'utilisation d'une base de données par défaut n'empêche pas l'accès aux autres bases. Par exemple, il est toujours possible d'accéder à la table `editeur` de la base `db2`, même après l'utilisation de `USE db1`

```
mysql62; USE db1;
mysql62; SELECT author_name,editor_name FROM author,db2.editor
           WHERE author.editor_id = db2.editor.editor_id;
```

La commande USE est fournie pour assurer la compatibilité avec Sybase.

7.18 FLUSH (vider les caches)

```
FLUSH flush_option [,flush_option]
```

La commande FLUSH permet d'effacer les caches internes de ***MySQL*** uses. Pour l'exécuter, il faut avoir les droits de rechargement (***reload***).

`flush_option` peut prendre les valeurs suivantes :

Ces commandes sont disponibles avec l'utilitaire `mysqladmin`, en utilisant les options `flush−hosts`, `flush−logs`, `reload` ou `flush−tables`.

7.19 KILL

```
KILL thread_id
```

```
KILL thread_id
```

Chaque connection à `mysqld` génére un thread séparé. La liste des threads est accessible avec la `SHOW PROCESSLIST` et la fonction `KILL thread_id`, permet de terminer immédiatement ces threads.

Il faut avoir les droits de ***process*** pour voir et terminer tous les threads. Sinon, seul les threads utilisateurs sont visibles et terminables.

Ces commandes sont disponibles avec l'utilitaire `mysqladmin` en utilisant les options `mysqladmin processlist` and `mysqladmin kill`.

7.20 SHOW (Informations sur les tables, colonnes,...)

```
   SHOW DATABASES [LIKE wild]
ou SHOW TABLES [FROM Nom_bdd] [LIKE wild]
ou SHOW COLUMNS FROM Nom_table [FROM Nom_bdd] [LIKE wild]
ou SHOW INDEX FROM Nom_table [FROM Nom_bdd]
ou SHOW STATUS
ou SHOW VARIABLES [LIKE wild]
ou SHOW PROCESSLIST
ou SHOW TABLE STATUS [FROM Nom_bdd] [LIKE wild]
```

`SHOW` fourni les caractéristiques des bases de données, tables et colonnes sur le serveur. Si l'option `LIKE wild` est utilisée, la chaîne `wild` peut utiliser les caractères spéciaux ``%'' et ``_''.

Il y a deux possibilités pour référencer une table : `Nom_bdd.Nom_table` ou `Nom_table FROM Nom_bdd` syntax. Les deux requêtes suivantes sont équivalentes :

```
mysql62; SHOW INDEX FROM maTable FROM maBase;
mysql62; SHOW INDEX FROM maBase.maTable;
```

`SHOW DATABASES` fait la liste des bases de donnés sur le serveur ***MySQL***. `mysqlshow` fournit les mêmes renseignements.

`SHOW TABLES` fait la liste des tables sur une base donnée. `mysqlshow Nom_bdd` fournit les mêmes renseignements.

Note: Si un utilisateur n'a aucun droit pour une table, la table n'apparaitra pas dans les `SHOW TABLES` ou `mysqlshow Nom_bdd`.

`SHOW COLUMNS` fait la liste des colonnes d'une table. Si les types des colonnes sont différens de ce qui ont été spécifiés à la création, avec la `CREATE TABLE`, il faut savoir que parfois, ***MySQL*** change spontanément le type de colonnes. 7.6.1 Modifications automatiques de type de colonne.

La commande `DESCRIBE` fournit les mêmes informations que `SHOW COLUMNS`. DESCRIBE.

`SHOW TABLE STATUS` (nouveauté de la version 3.23) fonctionne comme `SHOW STATUS`, mais fournit beaucoup plus d'informations sur chaque table. L'utilitaire `mysqlshow` avec l'option `--status Nom_bdd` effectue la même requête. Les colonnes suivantes sont renvoyées :

Name	Name of the table
Type	Type of table (NISAM, MyISAM or HEAP)
Rows	Number of rows
Avg_row_length	Average row length
Data_length	Length of the data file

Max_data_length	Max length of the data file
Index_length	Length of the index file
Data_free	Number of allocated but not used bytes
Auto_increment	Next autoincrement value
Create_time	When the table was created
Update_time	When the data file was last updated
Check_time	When one last run a check on the table
Create_options	Extra options used with `CREATE TABLE`
Comment	The comment used when creating the table (or some information why MySQL couldn't access the table information).

`SHOW FIELDS` est un synonyme de `SHOW COLUMNS` et `SHOW KEYS` est un synonyme de `SHOW INDEX` Ces commandes sont disponibles avec l'utilitaire `mysqlshow Nom_bdd Nom_table` ou `mysqlshow −k Nom_bdd Nom_table`.

`SHOW INDEX` retourne les informations dans un format proche de `SQLStatistics` sous ODBC. Les informations suivantes sont disponibles :

Table	Name of the table
Non_unique	0 if the index can't contain duplicates.
Key_name	Name of the index
Seq_in_index	Column sequence number in index, starting with 1.
Column_name	Column name.
Collation	How the column is sorted in the index. In ***MySQL***, this can have values `A` (Ascending) or `NULL` (Not sorted).
Cardinality	Number of unique values in the index. This is updated by running `isamchk −a`.
Sub_part	Number of indexed characters if the column is only partly indexed. `NULL` if the entire key is indexed.

SHOW STATUS fournit des informations sur le serveur (tout comme `mysqladmin extended−status`). Les informations fournies sont présentées ci−dessous. Le format et les nombres peuvent varier :

```
+−−−−−−−−−−−−−−−−−−−−−−−−+−−−−−−−−+
| Variable_name            | Value  |
+−−−−−−−−−−−−−−−−−−−−−−−−+−−−−−−−−+
| Aborted_clients          | 0      |
| Aborted_connects         | 0      |
| Created_tmp_tables       | 0      |
| Delayed_insert_threads   | 0      |
| Delayed_writes           | 0      |
| Delayed_errors           | 0      |
| Flush_commands           | 2      |
| Handler_delete           | 2      |
| Handler_read_first       | 0      |
| Handler_read_key         | 1      |
| Handler_read_next        | 0      |
| Handler_read_rnd         | 35     |
| Handler_update           | 0      |
| Handler_write            | 2      |
| Key_blocks_used          | 0      |
| Key_read_requests        | 0      |
| Key_reads                | 0      |
| Key_write_requests       | 0      |
| Key_writes               | 0      |
```

```
| Max_used_connections     | 1      |
| Not_flushed_key_blocks   | 0      |
| Not_flushed_delayed_rows | 0      |
| Open_tables              | 1      |
| Open_files               | 2      |
| Open_streams             | 0      |
| Opened_tables            | 11     |
| Questions                | 14     |
| Running_threads          | 1      |
| Slow_requêtes            | 0      |
| Uptime                   | 149111 |
+--------------------------+--------+
```

Les variables ci−dessus ont les significations suivantes :

`Aborted_clients`	Number of connections that has been aborted because the client has died without closing the connection properly.
`Aborted_connects`	Number of tries to connect to the MySQL server that has failed.
`Created_tmp_tables`	Number of implicit temporary tables that has been created while executing statements.
`Delayed_insert_threads`	Number of delayed insert handler threads in use.
`Delayed_writes`	Number of rows written with `INSERT DELAYED`.
`Delayed_errors`	Number of rows written with `INSERT DELAYED` for which some error occurred (probably `duplicate key`).
`Flush_commands`	Number of executed `FLUSH` commands.
`Handler_delete`	Number of requests to delete a row from a table.
`Handler_read_first`	Number of request to read first the row in a table.
`Handler_read_key`	Number of request to read a row based on a key.
`Handler_read_next`	Number of request to read next row in key order.
`Handler_read_rnd`	Number of request to read a row based on a fixed position.
`Handler_update`	Number of requests to update a row in a table.
`Handler_write`	Number of requests to insert a row in a table.
`Key_blocks_used`	The number of used blocks in the key cache.
`Key_read_requests`	The number of request to read a key block from the cache.
`Key_reads`	The number of physical reads of a key block from disk.
`Key_write_requests`	The number of request to write a key block to the cache.
`Key_writes`	The number of physical writes of a key block to disk.
`Max_used_connections`	The maximum number of connections that has been in use simultaneously.
`Not_flushed_key_blocks`	Keys blocks in the key cache that has changed but hasn't yet been flushed to disk.
`Not_flushed_delayed_rows`	Number of rows waiting to be written in `INSERT DELAY` queues.
`Open_tables`	Number of tables that are open.
`Open_files`	Number of files that are open.
`Open_streams`	Number of streams that are open (used mainly for logging)
`Opened_tables`	Number of tables that has been opened.
`Questions`	Number of questions asked from to the server.
`Running_threads`	Number of currently open connections.
`Slow_queries`	Number of requêtesthat has taken more than `long_query_time`
`Uptime`	How many seconds the server has been up.

Commentaires :

- Si `Opened_tables` est grand, alors `table_cache` est probablement trop petit.
- Si `key_reads` est trop grand, alors `key_cache` est probablement trop petit. Le taux d'accès au cache est calculé avec la `key_reads/key_read_requests`.
- Si `Handler_read_rnd` est grand, alors il y a probablement trop de requêtes qui obligent ***MySQL*** a scanner des tables entières, ou il y a des commandes join qui n'utilisent pas les clés à bon escient.

`SHOW VARIABLES` affiche quelques variables système de ***MySQL***. Cette commande est disponible avec l'utilitaire `mysqladmin variables` command. Si les valeurs par défaut ne conviennent pas, il faut faire les réglages au démarrage, dans la commande de `mysqld` . Les informations fournit sont les suivantes, et ressemble fort au tableau ci−dessous :

```
+−−−−−−−−−−−−−−−−−−−−−−+−−−−−−−−−−−−−−−−−−−−−−−−−−−+
| Variable_name          | Value                       |
+−−−−−−−−−−−−−−−−−−−−−−+−−−−−−−−−−−−−−−−−−−−−−−−−−−+
| back_log               | 5                           |
| connect_timeout        | 5                           |
| basedir                | /my/monty/                  |
| datadir                | /my/monty/data/             |
| delayed_insert_limit   | 100                         |
| delayed_insert_timeout | 300                         |
| delayed_queue_size     | 1000                        |
| join_buffer_size       | 131072                      |
| flush_time             | 0                           |
| key_buffer_size        | 1048540                     |
| language               | /my/monty/share/english/    |
| log                    | OFF                         |
| log_update             | OFF                         |
| long_query_time        | 10                          |
| low_priority_updates   | OFF                         |
| max_allowed_packet     | 1048576                     |
| max_connections        | 100                         |
| max_connect_errors     | 10                          |
| max_delayed_threads    | 20                          |
| max_heap_table_size    | 16777216                    |
| max_join_size          | 4294967295                  |
| max_sort_longueur      | 1024                        |
| max_tmp_tables         | 32                          |
| net_buffer_longueur    | 16384                       |
| port                   | 3306                        |
| protocol−version       | 10                          |
| record_buffer          | 131072                      |
| skip_locking           | ON                          |
| socket                 | /tmp/mysql.sock             |
| sort_buffer            | 2097116                     |
| table_cache            | 64                          |
| thread_stack           | 131072                      |
| tmp_table_size         | 1048576                     |
| tmpdir                 | /machine/tmp/               |
| version                | 3.23.0−alpha−debug          |
| wait_timeout           | 28800                       |
+−−−−−−−−−−−−−−−−−−−−−−+−−−−−−−−−−−−−−−−−−−−−−−−−−−+
```

Server parameters.

SHOW PROCESSLIST affiche la liste des processus en ligne. Cette commande est aussi disponible avec la `mysqladmin processlist`. Avec les droits de ***process***, tous les threads sont visibles. Sinon, seuls les threads utilisateurs sont visibles . `KILL. KILL.`

7.21 EXPLAIN (Détails sur SELECT)

```
EXPLAIN SELECT select_options
```

L'option EXPLAIN force ***MySQL*** à expliquer la façon avec laquelle il va traiter la requête SELECT, en détaillant les opérations de regroupement.

Avec ces informations, il est possible de determiner les tables qui requièrent des indexes pour accélerer les SELECT, ainsi que l'ordre optimal dans lequel les tables reseront regroupées. Pour obliger l'optimiseur à respecter l'ordre de regroupement, il suffit d'ajouter l'option STRAIGHT_JOIN.

Pour les regroupements complexes, EXPLAIN retourne une ligne d'informations pour chaque table utilisée par la commande SELECT. Les tables sont listées dans l'ordre de lecture. ***MySQL*** résoud les regroupements en utilisant une méthode d'aggrégation en un seul passage. Cela signifie que ***MySQL*** lit une ligne de la première table, puis il recherche les lignes correspondantes dans la seconde table, et ainsi de suite jusqu'à la dernière table. Quand toutes les tables ont été traitées, il retourne la ligne selectionnée, et remonte progressivement jusqu'à la table initiale. Puis, la ligne suivante est selectionnée, et le processus continue avec la ligne suivante.

La réponse de EXPLAIN inclus les colonnes suivantes :

- `table` La table qui est utilisée par la commande.
- `type` Le type de regroupement. Le détails sur les différents types est donnés plus bas.
- `possible_keys` Les `possible_keys` indiques quels index ***MySQL*** peut utiliser pour rechercher des lignes dans la table. Si la colonne est vide, il n'y a pas d'index. Dans ce cas, il est surement possible d'améliorer les performances de la requête en éxaminant la clause `WHERE` pour voir quelles colonnes sont utilisées, et quelles colonnes mériteraient un indexage. Dans ce cas, il suffit de créer l'index adéquat, et de reverifier la requête avec `EXPLAIN` . Pour voir quels sont les index disponibles pour une table, il faut utiliser la commande `SHOW INDEX FROM Nom_table`.
- `key` La colonne clé indique quelle clé ***MySQL*** a décidé d'utiliser. La clé est `NULL`si aucun index n'est choisi.
- `key_len key_len` indique la longueur de la clé que a décidé d'utiliser. La longueur sera `NULL` si la clé est `NULL`.
- `ref ref` est le ou les numéros de colonne ou constantes utilisée avec la clé pour rechercher les lignes dans la table.
- `rows rows` indique le nmbre de ligne que ***MySQL*** doit examiner pour éxécuter la requête.
- `Extra` Si la colonne inclus le texte `Only index` , cela signifie que les informaions sont renvoyées par la table en utilisant uniquement l'index. Généralement, c'est beaucoup plus rapide que de scanner la table entière. Si cette colonne contient le texte `where used`, cela signifie que la clause `WHERE` a été utilisée pour restreindre le nombre de ligne à retourner au client.

Voici maintenant la liste des différents types de regroupement, du plus efficace au moins efficace :

- `system` La table n'a qu'une seule ligne (= table système). C'est un cas spécial de regroupement de type `const`.
- `const` La table a au maximum une ligne à traiter, qui sera lue au début de la requête. Etant donné qu'il n'y a qu'une seule ligne, les valeurs de cette ligne peuvent être considérées comme des constantes pour l'optimisateur. Les tables de type `const` sont extrêmement rapide à lire!
- `eq_ref` Une ligne sera lue de cette table pour chaque combinaisons de ligne des tables précédentes. C'est le meilleur type de regroupement possible, en dehors du type `const`. Il sera utilissé lorsque toutes les colonnes d'un index sont utilisées par un regroupement, et que l'index est `UNIQUE` ou `PRIMARY KEY`.
- `ref` Toutes les lignes qui correspondent aux valeurs de l'index seront lues dans cette table, pour chaque combinaison de lignes des tables précédentes. `ref` est utilisé si le regroupement utilise un préfixe comme clé, ou si la clé n'est pas `UNIQUE` ou `PRIMARY KEY`(en d'autres termes, si le regroupement ne peut pas sectionner une ligne unique à partir de la clé). Si la clé qui est utilisée ne rassemble que très peu de lignes, le regroupement est bon.
- `range` Seules les lignes dans l'intervalle considéré seront renvoyée, en utilisant un index pour selectionner les lignes. La colonne `ref` indiquera quelles index sera utilisé.
- `index` Ce type est indentique à `ALL`, sauf que seul l'index est scanné. C'est généralement plus rapide que le type `ALL`, car un fichier d'index est généralement plus petit que le fichier de données.
- `ALL` Une recherche sur la table complète va être faite, pour chaque combinaison des tables précédentes. C'est généralement très mauvais si la première table n'est pas marquée `const`, et encore plus mauvais dans les autres cas. On peut éviter d'utiliser ce mode de recherche en ajoutant des index, afin de transformer les lignes en constantes.

Un bon critère pour evaluer l'efficacité d'un regroupement est de multiplier toutes les valeurs dans la colonne `rows` d'une requête avec EXPLAIN. Cela mesure approximativement le nombre de ligne que ***MySQL*** doit examiner pour résoudre la requête. Ce nombre sera aussi utilisé pour restreindre la taille du regroupement, avec `max_join_size`.

L'exemple suivant montre comme une clause peut être optimisée progressivement gr ce aux informations fournies par EXPLAIN. On supposera que l'on souhaite exécuter la commande SELECTci−dessous, et qu'on l'examine avec EXPLAIN:

```
EXPLAIN SELECT tt.TicketNumber, tt.TimeIn,
            tt.ProjectReference, tt.EstimatedShipDate,
            tt.ActualShipDate, tt.ClientID,
            tt.ServiceCodes, tt.RepetitiveID,
            tt.CurrentProcess, tt.CurrentDPPerson,
            tt.RecordVolume, tt.DPPrinted, et.COUNTRY,
            et_1.COUNTRY, do.CUSTNAME
        FROM tt, et, et AS et_1, do
        WHERE tt.SubmitTime IS NULL
            AND tt.ActualPC = et.EMPLOYID
            AND tt.AssignedPC = et_1.EMPLOYID
            AND tt.ClientID = do.CUSTNMBR;
```

Pour cet exemple, on supposera par hypothèse :

- Les colonnes comparées sont déclarées comme suit :

Table	*Column*	*Column type*
tt	ActualPC	CHAR(10)
tt	AssignedPC	CHAR(10)
tt	ClientID	CHAR(10)
et	EMPLOYID	CHAR(15)
do	CUSTNMBR	CHAR(15)

- Les tables ont les index suivants :

Table	*Index*
tt	ActualPC
tt	AssignedPC
tt	ClientID
et	EMPLOYID (primary key)
do	CUSTNMBR (primary key)

- Les valeurs de tt.ActualPC n'ont pas encore été assignées.

Au démarrage, avant la moindre optimisation, la clause produit la réponse suivante :

```
table type possible_keys               key  key_len ref  rows  Extra
et    ALL  PRIMARY                     NULL NULL    NULL 74
do    ALL  PRIMARY                     NULL NULL    NULL 2135
et_1  ALL  PRIMARY                     NULL NULL    NULL 74
tt    ALL  AssignedPC,ClientID,ActualPC NULL NULL    NULL 3872
      range checked for each record (key map: 35)
```

Etant donné que le type de regroupement est ALL pour toutes les tables, ces informations indique que ***MySQL***fait un regroupement sur toutes les tables! Cela va prendre un temps énorme, vu le nombre de lignes à étudier dans chaque table. Pour le cas présent, cela représente 74 * 2135 * 74 * 3872 = 45,268,558,720 rows. Et encore, ces tables pourraient être encore plus grosses...

Un des problèmes posés ici est que ***MySQL*** ne peut pas encore (encore) utiliser d'index pour des colonnes déclarées de manière différentes. Dans l'exemple, VARCHAR et CHAR sont identiques, à moins qu'ils ne soient déclarés sur des longueurs différentes. Or, tt.ActualPC est de type CHAR(10) et et.EMPLOYID est de type CHAR(15) : les longueur ne concordent pas.

Pour consolider la base, on utilise la commande ALTER TABLE pour rallonger le champs ActualPC de 10 caractères à 15 caractères:

```
mysql62; ALTER TABLE tt MODIFY ActualPC VARCHAR(15);
```

Maintenant, tt.ActualPC et et.EMPLOYID sont tous les deux de type VARCHAR(15). L'éxecution de la commande EXPLAIN produit maintenant le résultat suivant :

```
table type   possible_keys   key     key_len ref         rows    Extra
tt    ALL    AssignedPC,ClientID,ActualPC NULL NULL NULL 3872    where used
do    ALL    PRIMARY         NULL    NULL    NULL        2135
      range checked for each record (key map: 1)
et_1  ALL    PRIMARY         NULL    NULL    NULL        74
      range checked for each record (key map: 1)
et    eq_ref PRIMARY         PRIMARY 15      tt.ActualPC 1
```

Ce n'est pas parfait, mais c'est nettement mieux (le produit des colonnes rows a été réduit d'un facteur 74). Cette version s'éxecute maintenant en quelques secondes.

Une autre amélioration peut être apportée en éliminant les disparités de longueur entre les colonnes tt.AssignedPC et et_1.EMPLOYID, et d'autres part , tt.ClientID et do.CUSTNMBR comparisons:

```
mysql62; ALTER TABLE tt MODIFY AssignedPC VARCHAR(15),
                     MODIFY ClientID   VARCHAR(15);
```

Maintenant EXPLAIN produit maintenant le résultat suivant :

```
table type   possible_keys   key     key_len ref            rows     Extra
et    ALL    PRIMARY         NULL    NULL    NULL           74
tt    ref    AssignedPC,ClientID,ActualPC ActualPC 15 et.EMPLOYID 52 where used
et_1  eq_ref PRIMARY         PRIMARY 15      tt.AssignedPC  1
do    eq_ref PRIMARY         PRIMARY 15      tt.ClientID    1
```

C'est déjà très bien.

Le problème final est que, par défaut, ***MySQL*** suppose que les valeurs dans la colonne tt.ActualPC sont réparties uniformément. Or, ce n'est pas le cas de la table tt. Heureusement, il est facile de le préciser à ***MySQL*** :

```
shell62; isamchk −−analyze PATH_TO_MYSQL_DATABASE/tt
shell62; mysqladmin refresh
```

Le résultat est maintenant parfait, et maintenant EXPLAIN produit maintenant le résultat suivant :

```
table type   possible_keys   key     key_len ref            rows    Extra
tt    ALL    AssignedPC,ClientID,ActualPC NULL NULL NULL    3872    where used
et    eq_ref PRIMARY         PRIMARY 15      tt.ActualPC    1
et_1  eq_ref PRIMARY         PRIMARY 15      tt.AssignedPC  1
do    eq_ref PRIMARY         PRIMARY 15      tt.ClientID    1
```

On peut noter que la colonne rows est une estimation de la part de l'optimisateur de regroupement de ***MySQL*** : pour optimiser une commande, il faudrait maintenant vérifier qu'elle a des chiffres proche de la vérité. Si non, il faudrait améliorer les performances avec la clause STRAIGHT_JOIN dans la commande SELECT, et essayer de proposer différents ordres pour la liste des tables .

7.22 DESCRIBE (Informations sur les colonnes)

```
{DESCRIBE | DESC} Nom_table {Nom_col | wild}
```

DESCRIBE fournit des informations à propos des colonnes d'une table. Nom_col peut être un nom de colonne, ou une chaîne qui contenant le caractère spécial SQL ``%'' et ``_''.

DESCRIBE fait la liste des colonnes d'une table. Si les types des colonnes sont différens de ce qui ont été spécifiés à la création, avec la CREATE TABLE, il faut savoir que parfois, ***MySQL*** change spontanément le type de colonnes. 7.6.1 Modifications automatiques de type de colonne.

La commande USE est fournie pour assurer la compatibilité avec Oracle.

La commande SHOW fournit les mêmes informations. SHOW.

7.23 LOCK TABLES/UNLOCK TABLES

```
LOCK TABLES Nom_table [AS alias] {READ | [LOW_PRIORITY] WRITE}
            [, Nom_table {READ | [LOW_PRIORITY] WRITE} ...]
...
UNLOCK TABLES
```

LOCK TABLES verrouille une table dans le thread courant. UNLOCK TABLES ouvre tous les verrous posé par le thread courant. Toutes les tables verrouillé par un thread sont automatiquement déverrouillée quand le thread émet un autre LOCK TABLES, ou à la fin de la connexion au serveur.

Si un thread obtiens le verrou de lecture (READ) sur une table, le thread (et tous les autres threads) ne peut que lire dans la table. Si un thread obtiens e verrou de lecture (READ) sur une table, le thread qui a le verrous est le seul à pouvoir lire ou écrire dans la table.

Les autres threads attendent (sans limite) que le verrous se libère.

Le verrous d'écriture a une priorité supérieure au verrou de lecture, afin que les processus de mise à jour puisse se faire dès que possible. Cela signifie que si un thread obtiens un verrou de lecture, et qu'un autre thread obtiens un verrou d'écriture, alors le thread au verrou de lecture devra attendre la libération du verrou d'écriture. Il est possible d'utiliser des verrous d'écriture de basse priorité (LOW_PRIORITY WRITE), mais il faut être sur qu'il y aura un moment ou aucun thread ne sera en train de lire la table.

Lors de l'utilisation de la commande LOCK TABLES, il faut verrouiller toutes les tables qui vont être utilisées. Si il y a des alias dans une requête, il faut aussi avoir les verrous pour les alias! Cette politique assure que la table ne se verrouille jamais, sans pouvoir être déverrouillée.

Il ne faut jamais verrouiller une table qui va accepter une insertion reportée (INSERT DELAYED). Car, dans ce cas, l'insertion sera faite dans un autre thread, qui n'aura pas le verrou.

Généralement, il n'y a pas besoin de verrouiller les tables, car les mise à jour UPDATE sont atomiques : aucun autre thread ne peut interférer avec la commande en cours d'éxécution. Il y a toutes fois, quelques cas où il est bon de verrouiller une table :

- Si un grand nombre d'opération vont être menée sur un bon nombre de table, il est plus rapide de verrouiller les tables utilisées.

L'inconvénient, bien sur, est qu'aucun autre thread ne pourra accèder aux informations, ni les modifier.

- ***MySQL*** ne supporte pas d'environnement transactionnel, donc il faut absolument verrouiller une table, pour s'assurer qu'au autre thread n'intervient entre une commande `SELECT` et une commande `UPDATE` . L'exemple ci−dessous montre comment exécuter une transaction :

```
mysql62; LOCK TABLES trans READ, customer WRITE;
mysql62; select sum(value) from trans where customer_id= some_id;
mysql62; update customer set total_value=sum_from_previous_statement
           where customer_id=some_id;
mysql62; UNLOCK TABLES;
```

Sans la commande `LOCK TABLES`, il se peut qu'un autre thread insère une nouvelle ligne dans la table `trans` entre les deux commandes `SELECT` et `UPDATE` .

En utilisant des modifications incrémentales (`UPDATE customer SET value=value+new_value`) ou avec la commande `LAST_INSERT_ID()`, on peut généralement éviter l'utilisation des `LOCK TABLES`.

Il est aussi possible de résoudre quelques cas en utilisant les verrous utilisateurs `GET_LOCK()` et `RELEASE_LOCK()`. Ces verrous sont sauvés dans une table du serveur, et programmé avec `pthread_mutex_lock()` et `pthread_mutex_unlock()` pour plus de rapidité.n `Miscellaneous functions`. 10.11 Comment MySQL verrouille les tables.

7.24 SET OPTION

```
SET [OPTION] SQL_VALUE_OPTION= value, ...
```

`SET OPTION` selectionne différentes options qui affecteront le mode opératoire du client ou du serveur. Toute option reste valable jusqu'à la fin de la session, ou jusqu'à la prochaine modification.

- `CHARACTER SET character_set_name | DEFAULT` Cette option permet de choisir la table des caractères utilisée par ***MySQL***. Actuellement, la seule option possible est `cp1251_koi8`, mais il est très simple d'ajouter de nouvelles tables en éditant le fichier ``sql/convert.cc'' dans le code source de ***MySQL*** . La table par défaut peut être rappelée en utilisant la valeur `DEFAULT`. Il faut noter que la syntaxe pour choisir la table de caractère est différente des autres.
- `PASSWORD = PASSWORD('some password')` Choisit un nouveau mot de passe pour l'utilisateur courant. Tout utilisateur non−anonyme peut changer son mot de passe!
- `PASSWORD FOR user = PASSWORD('some password')` Assigne un nouveau mot de passe pour un utilisateur du serveur courant. Seul, un utilisateur avec des droits d'accès à `mysql` database peut le faire. L'utilisateur modifié doit être désigné par `utilisateur@nom_hote` , avec `utilisateur` et `nom_hote` qui prennent les valeurs qui apparaissent dans la table`mysql.user`, sous les colonnes `User` et `Host` columns of the table entry. Par exemple, si il existe une ligne avec `User` et `Host` qu valent respectivement `'bob'` et `'%.loc.gov'`, il faudra écrire:

```
mysql62; SET PASSWORD FOR bob@"%.loc.gov" = PASSWORD("newpass");
```

- `SQL_BIG_TABLES = 0 | 1` Si mis à un, toutes les tables temporaires sont stockés sur le disque dur, plutôt qu'en mémoire. Cela rend le processus un peu plus lent, mais il génère pas d'erreur du type `The table Nom_table is full` (la table Nom_table est pleine), si de grosses commandes `SELECT` demandes de grandes tables temporaires. Par défaut, cette option est à 0.
- `SQL_BIG_SELECTS = 0 | 1` Si mis à `1`, ***MySQL*** annulera une commande `SELECT` qui va prendre un temps trop long. Ceci est très utile quand une clause `WHERE` complexe a été spécifiée. Une requête trop long est une commande `SELECT` qui va avoir à étudier plus de `max_join_size` rows lignes. Par défaut, cette valeur est à 0 (Toutes les commandes `SELECT` autorisées).
- `SQL_LOW_PRIORITY_UPDATES = 0 | 1` Si mis à `1`, toutes les commandes `INSERT`, `UPDATE` et `DELETE` attendent qu'il n'y ait plus de commande `SELECT` en attente sur la table affectée.
- `SQL_SELECT_LIMIT = value | DEFAULT` Le nombre maximal de ligne à retourner dans une commande `SELECT`. Si une commande `SELECT` a une clause de limite `LIMIT`, `LIMIT` est prioritaire sur `SQL_SELECT_LIMIT`. La valeur par défaut pour cette option est "sans limite". Si la limite a été changée, il est toujours possible de restaurer la configuration initiale avec `DEFAULT`.
- `SQL_LOG_OFF = 0 | 1` Si mis à `1`, aucun historique ne sera transmis au client, si le client a les droits de ***process*** . Ceci n'affecte pas l'historique de mise à jour.
- `SQL_LOG_UPDATE = 0 | 1` Si mis à `0`, aucun historique de modification ne sera tenu, si le client a les droits de ***process*** privilege. Ceci n'affecte pas l'historique du client.
- `TIMESTAMP = timestamp_value | DEFAULT` Met à l'heure le client. Cette fonction est généralement utilisée pour fixer la valeur initiale du timestamp, lors de l'utilisation de l'historique pour recréer des lignes.
- `LAST_INSERT_ID = #` Fixe la valeur qui sera retournée par la prochaine fonction `LAST_INSERT_ID()`. Elle est stockée dans l'historique de modification.
- `INSERT_ID = #` Fixe la prochaine valeur à utiliser lors d'une insertion dans une table avec une colonne de type `AUTO_INCREMENT` value. Cela sert surtout avec l'historique de modifications.

7.25 GRANT et REVOKE

```
GRANT priv_type [(column_list)] [, priv_type [(column_list)] ...]
    ON {Nom_table | * | *.* | Nom_bdd.*}
    TO user_name [IDENTIFIED BY 'password']
        [, user_name [IDENTIFIED BY 'password'] ...]
    [WITH GRANT OPTION]
REVOKE priv_type [(column_list)] [, priv_type [(column_list)] ...]
    ON {Nom_table | * | *.* | Nom_bdd.*}
    FROM user_name [, user_name ...]
```

GRANT est implémenté depuis la version 3.22.11de ***MySQL***. Pour les ancienne versions, la commande GRANT ne fait rien.

Les commandes GRANT et REVOKE permettent aux administrateurs système de donner et enlever des droits aux utilisateurs :

- ***niveau général (Global level) :*** Les droits de niveau général s'applique à toutes les bases de données du serveur. Ces droits sont stocké dans la table `mysql.user`.
- ***Niveau base de données (Database level):*** Les droits de niveau base de données s'applique aux tables d'une base de données. Ces droits sont stockés dans les tables `mysql.db` et `mysql.host`.
- ***Niveau table (Table level) :*** Les droits de niveau table s'appliquent aux colonnes d'une table donnée. apply to all columns in a given table. Ces droits sont stockés dans la table `mysql.tables_priv` .
- ***Niveu colonne (Column level) :*** Les droits de niveau colonne s'appliquent à une colonn donnée d'une table donnée. to single columns in a given table. Ces droits sont stockés dans la table `mysql.columns_priv` .

Les commandes GRANT et REVOKE peuvent s'appliquer aux droits suivants, précisés dans `priv_type`:

```
ALL PRIVILEGES      FILE                RELOAD
ALTER               INDEX               SELECT
CREATE              INSERT              SHUTDOWN
DELETE              PROCESS             UPDATE
DROP                REFERENCES          USAGE
```

ALL est un synonyme de for `ALL PRIVILEGES`. REFERENCES n'est pas encore implémenté. USAGE est actuellement synonyme de ``aucun droits".Il peut être utilisé pour créer un utilisateur sans droits.

Pour enlever le droit de ***grant*** d'un utilisateur, il faut utiliser la valeur suivante dans `priv_type` :

```
REVOKE GRANT OPTION ON ... FROM ...;
```

Les seules valeurs de `priv_type` qu'il est possible de spécifier sont SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, GRANT, INDEX et ALTER.

Les seules valeurs de `priv_type` qu'il est possible de spécifier pour une colonne (c'est à dire, en utilisant la clause `column_list`) sont SELECT, INSERT et UPDATE.

Il est possible de donner les droits généraux en utilisant la syntaxe `ON *.*` . Il est possible de donner des droits sur une base de données en utilisant la syntaxe `ON Nom_bdd.*` syntax. Si il y a une base de données courante, et que l'on spécifie `ON *,` cela revient à donner tous les droits pour la base de données courante . (***Attention:*** Si il n'y a pas de base de données courante, cela revient à donner des droits sur le serveur entier!)

Afin de pouvoir donner des droits à des utilisateurs d'hotes divers et variés, ***MySQL*** accepte la spécification de l'utilisateur sous la forme `user@host`. Pour pouvoir spécifier des noms d'utilisateur ou d'hotes ayant des

caractères spéciaux dans le nom (comme ``−''), il suffit d'entourer le nom de l'utilisateur ou de l'hote de guillemets simples (i.e., 'test−user''test−hostname').

Les noms d'hotes peuvent avoir des caractères jokers : Par exemple, user@"%.loc.gov" s'applique à tous les utilisateurs user pour tous les hotes du domaine loc.gov et user@"144.155.166.%" s'appliquera à tous les user pour des hotes dans le domaine 144.155.166 .

La forme simplifié user est un synonyme de user@"%". ***Note:*** pour autoriser des utilisateurs anonymes sur le serveur ***MySQL*** (ce qui est le comportement par défaut), il faut aussi ajouter les utilisateurs locaux, gr ce à la forme user@localhost car sinon, la ligne pour l'utilisateur anonyme dans la table mysql.user sera utilisée quand l'utilisateur essaiera de se connecter à ***MySQL*** depuis la machine locale! Les utilisateurs anonymes sont définis en insérant une ligne User='' dans la table mysql.user . On peut vérifier si cela s'applique en exécutant la ligne suivante :

```
mysql62; SELECT Host,User FROM mysql.user WHERE User='';
```

Pour le moment, la commande GRANT n'accepte que des hotes, tables, bases et colonnes dont le nom ne dépasse pas 60 caractères. Un nom d'utilisateur peut aller jusqu'à 16 caractères.

Les privilèges pour une table ou une colonne sont constitués du OU logique de chacun des droits. Par exemple, si la table mysql.user précise que un utilisateur a un droit de ***select*** gloabal, cela ne peut pas être interdit par une entrée dans une base, une table ou une colonne.

Les droits pour une colonne sont calculés comme suit :

```
global privileges
OR (database privileges AND host privileges)
OR table privileges
OR column privileges
```

Dans la plupart des cas, les droits sont donnés aux utilisateurs avec les niveaux de droits, ce qui simplifie grandement la vie.

Si des droits sont donnés à un couple user/hostname qui n'existe pas dans la table mysql.user, une entrée est insérée dans la table des droits, et elle reste valide jusqu'à ce qu'une commande DELETE l'efface. En d'autres termes, GRANT peut créer un utilisateur, mais REVOKE ne l'effacera pas. Il faut le faire explicitement avec DELETE.

A partir de ***MySQL*** 3.22.12 , lors de la création d'un nouvel utilisateur ou avec les droits globaux, un mot de passe sera affecté avec la clause IDENTIFIED BY . Si l'utilisateur à déjà un mot de passe, il sera remplacé par le nouveau.

Attention: Lors de la création d'un utilisateur, il aucun mot de passe n'est spécifiée, cet utilisateur n'aura pas de mot de passe. C'est très imprudent.

Les mots de passe peuvent être assignés et modifiés avec la commande : SET PASSWORD . SET OPTION.

Lors de l'attribution de droit de niveau base , une entrée est ajoutée dans la table mysql.db, si nécessaire. Quand toutes les privilèges poue la base de données sont supprimés avec REVOKE, l'entrée est effacée.

Si un utilisateur n'a aucun droit sur uen table, la table n'est pas affichée quand l'utilisateur fait une requête sur cette table. La table ne sera même pas visible avec une commandes SHOW TABLES.

La clause `WITH GRANT OPTION` donne à l'utilisateur la capacité de donner à d'autres utilisateurs des droits, d'un niveau égal à ceux qu'il a déjà. Il faut être très prudent quand on attribue ce droit, car deux utilisateurs avec des droits différents peuvent rassembler leurs droits.

Il faut bien savoir que lorsqu'on donne à un utilisateur le droit de grant, tous les droits que possède cet utilisateur sont transmissible par cet utilisateur. Supposons que l'on ait donné des droits d'insertion à un utilisateur. Si on lui ajoute le droit de select sur une base, et qu'on lui ajoute en plus `WITH GRANT OPTION`, l'utilisateur peut donner les droits de selection et d'insertion à tout autre utilisateur. Et si on lui ajoute encore le droit de update, il pourra aussi donner ce droit.

Il ne vaut mieux pas donner des droits de alter à un utilisateur normal. Dans ce cas, l'utilisateur peut essayer de pirater le système en renommant les tables de droits du système.

Lors de l'utilisation des droits de table ou de colonne pour un utilisateur, le serveur examine les droits de table et de colonne pour tous les utilisateurs, et cela ralentit légérement ***MySQL***.

Quand `mysqld` démarre, tous les droits sont chargés en mémoire. Les droits de table, base et colonnes prennent effet aussitôt, et les droits d'utilisateur prennent effet à la première connexion. Les modifications des tables de droits sont faites avec les fonctions `GRANT` et `REVOKE` et sont repercutées par le serveur immédiatement.Si les tables de droits sont modifiées à la main (avec `INSERT`, `UPDATE`, etc.), il faut exécuter une commande `FLUSH PRIVILEGES` ou lancer l'utilitaire `mysqladmin flush−privileges` pour faire prendre en compte des nouveaux droits.

Les différences les plus notables entre ANSI SQL et ***MySQL*** pour la commande `GRANT` sont:

- ANSI SQL n' pas de droit de nvieau global ou base de données, et ANSI SQL ne supporte pas tous les types de droits de ***MySQL***.
- Lors de l'effacement d'une table en ANSI SQL, tous les privilèges de cette table sont révoqués. Lors de la révocation d'un droit avec ANSI SQL, tous les privilèges qui ont été octroyés avec ce droits sont aussi révoqués. Avec ***MySQL***, les droits peuvent être abandonnés uniquement avec la commande `REVOKE` ou en manipulant (avec précautions) les tables de droits.

7.26 CREATE INDEX

```
CREATE [UNIQUE] INDEX Nom_indexON Nom_table (Nom_col[(longueur]),... )
```

The `CREATE INDEX` n'est disponible qu'à partir de la version 3.22. `CREATE INDEX` est un raccourci de `ALTER TABLE` qui crée des index. `ALTER TABLE`

Généralement, il est possible de créer des tous les index d'une table au moment de la création de la table, avec `CREATE TABLE`. `CREATE TABLE`CREATE INDEX permettra alors d'ajouter de nouveaux index.

Une liste de nom de colonne de format `(col1,col2,...)` créer un index de multiples colonnes. Les index sont formés en concaténant les différentes valeurs en une ligne.

Pour les valeurs de type `CHAR` et `VARCHAR` , les index peuvent ne prendre en compte qu'une partie de la colonne, en précisant `Nom_col(longueur)` . (Avec les types `BLOB` et `TEXT`, cette longueur est obligatoire.). La commande suivante montre comment créer un index sur les 10 premiers caractères d'une colonne :

```
mysql62; CREATE INDEX part_of_name ON customer (name(10));
```

Etant donné que la plus part des mots diffèrent les uns des autres dans les 10 premières lettres, l'index crée ne

devrait pas être moins efficace que la colonne, tout en étant nettement plus rapide. Faire des index à valeur partiel permet de réduire la taille des index, et d'accélérer les opération de tris et d'insertion.

Il faut noter que l'on peut ajouter à un index une colonne qui accepte les types NULL, que depuis la version 3.23.2 de ***MySQL***. De même pour les colonnes de type BLOB/TEXT. Cela impose l'utilisation du format de table MyISAM.

7.27 DROP INDEX

```
DROP INDEX Nom_index
```

The DROP INDEX n'est disponible qu'à partir de la version 3.22. DROP INDEX est un raccourci de ALTER TABLE qui efface des index. ALTER TABLE ALTER TABLE.

7.28 Commentaires

Le serveur ***MySQL*** accepte les commentaires qui commencent par # jusqu'à la fin de la ligne, et les commentaires multi−lignes de type C/C++ /*.......... */:

```
mysql62; select 1+1;     # This comment continues to the end of line
mysql62; select 1 /* this is an in−line comment */ + 1;
mysql62; select 1+
/*
this is a
multiple−line comment
*/
1;
```

Bien que le serveur comprennent les commentaires multi lignes, il y a quelques restrictions :

- Le guillemets, simple et doubles, sont considés comme indiquant le début d'une chaîne, même à l'intérieur d'un commentaire. Si le guillement n'est pas doublé, l'analyseur ne réalise par que le commentaires est terminé. Sous mysql , l'invite de commande passe de mysql> à '> or ">.
- Un point virgule est considéré comme une fin de commande SQL, et tout ce qui est après est la prochaine commande.

Ces limitations s'applique aussi bien lors de l'utilsation de mysql que lors de la lecture d'un fichier dans une table.

MySQL n'accepte pas les commentaires de type ``--'' ANSI SQL. 5.3.7 `--' comme début de commentaire.

7.29 CREATE FUNCTION/DROP FUNCTION

```
CREATE FUNCTION function_name RETURNS {STRING|REAL|INTEGER}
       SONAME shared_library_name
DROP FUNCTION function_name
```

Une fonction définie par l'utilisateur est un bon moyen d'ajouter de nouvelles fonctionnalités à ***MySQL*** avec de nouvelles fonctions natives, comme par exemple ABS() et CONCAT().

`CREATE FUNCTION` sauve le nom de la fonction, le type et le point d'entrée de la fonction dans la table système `mysql.func` . Il faut avoir les droit ***insert*** et ***delete*** pour pouvoir créer et effacer des fonctions.

Toutes les fonctions actives sont rechargées à chaque démarrage du serveur, à moins de lancer `mysqld` avec l'option `−−skip−grant−tables` . Dans ce cas, l'initialisation des fonctions utilisateurs est oubliée, et les fonctions sont inutilisables. (Une fonction active est une fonction créée par `CREATE FUNCTION` et pas effacée avec `DROP FUNCTION`.)

7.30 Mots reservés par MySQL

Un problème commun est la création d'une table avec des noms de colonne qui sont aussi des nom de type de données, ou de fonction natives ***MySQL***. Ceci est parfaitement possible (par exemple, `ABS` peut être un nom de colonne), mais aucun espace n'est autorisé entre le nom d'une fonction est la parenthèse ouvrante lorsque ces noms sont utilisé comme des fonctions .

Les mots suivants sont explicitement reservés par ***MySQL***. La plus part sont interdits pas ANSI SQL92 comme nom de colonne ou nom de table (par exemple, group). Quelques uns sont reservés par ***MySQL*** qui en a besoin pour utiliser un analyseur syntaxique `yacc`

Les noms suivants (issus de la table ci−dessus) sont interdits pas ANSI SQL, mais acceptés apr ***MySQL*** comme nom de table/colonne. Ceci, car ces noms sont très courants, et de nombreuses personnes les utilisent déjà :

action	add	all	alter
after	and	as	asc
auto_increment	between	bigint	bit
binary	blob	bool	both
by	cascade	char	character
change	check	column	columns
constraint	create	cross	current_date
current_time	current_timestamp	data	database
databases	date	datetime	day
day_hour	day_minute	day_second	dayofmonth
dayofweek	dayofyear	dec	decimal
default	delete	desc	describe
distinct	distinctrow	double	drop
escaped	enclosed	enum	explain
exists	fields	first	float
float4	float8	foreign	from
for	full	function	grant
group	having	hour	hour_minute
hour_second	ignore	in	index
infile	insert	int	integer
interval	int1	int2	int3
int4	int8	into	if

is	join	key	keys
last_insert_id	leading	left	like
lines	limit	load	lock
long	longblob	longtext	low_priority
match	mediumblob	mediumtext	mediumint
middleint	minute	minute_second	month
monthname	natural	numeric	no
not	null	on	option
optionally	or	order	outer
outfile	partial	password	precision
primary	procedure	processlist	privileges
quarter	read	real	references
rename	regexp	reverse	repeat
replace	restrict	returns	rlike
second	select	set	show
smallint	soname	sql_big_tables	sql_big_selects
sql_select_limit	sql_low_priority_updates	sql_log_off	sql_log_update
straight_join	starting	status	string
table	tables	terminated	text
time	timestamp	tinyblob	tinytext
tinyint	trailing	to	use
using	unique	unlock	unsigned
update	usage	values	varchar
variables	varying	varbinary	with
write	where	year	year_month
zerofill			

Les valeurs suivantes issues de la table ci dessus) sont interdites par ANSI SQL, mais autorisées par ***MySQL*** comme nom de table ou de colonne. Ceci, car ce sont des noms naturels, et de nombreuses personnes les utilisent déjà.

- ACTION
- BIT
- DATE
- ENUM
- NO
- TEXT
- TIME
- TIMESTAMP

8 Exemple MySQL

Ce chapitre est une introduction à ***MySQL*** qui montre comment utiliser le client ***mysql*** pour créer et utiliser une base de données simple. ***mysql*** est un client (parfois appelée ``terminal'' ou aussi ``moniteur'') qui permet à un utilisateur de se connecter à un serveur ***MySQL***, de lancer quelques requêtes, et de voir les résultats. ***mysql*** peut aussi être lancé en mode automatique, en lui précisant un fichier qui contient les commandes à exécuter. Cette présentation de ***mysql*** couvre les deux aspects.

Pour avoir la liste des options disponibles sur ***mysql***, il suffit d'utiliser l'option : ***--help.***

```
shell62; mysql --help
```

Ce chapitre supposera que ***mysql*** est installé sur votre machine, et qu'un serveur ***MySQL*** est accessible. Si ce n'est pas le cas, contactez votre administrateur ***MySQL*** (Si vous etes l'administrateur, vous aurez certainement besoin de consulter d'autres sections de ce manuel).

Ce chapitre couvre la constitution et l'utilisation d'une base de données. Si vous êtes simplement interessé par la lecture de bases de données déjà existantes, vous pouvez éviter les premières sections qui montre la création d'une base de données et de tables.

Etant donné que ce chapitre n'est qu'un exemple d'introduction, de nombreux détails sont laissés de coté. N'hésitez pas à vous reportez aux autres sections du manuel, pour toute information complémentaire.

8.1 Connection et déconnection du serveur

Pour se connecter au serveur ***MySQL***, il vous faut un nom d'utilisateur, et, le plus probablement, un mot de passe. Si le serveur tourne sur une autre machine, il vous faudra aussi un nom d'hote. Contactez l'administrateur de la machine pour connaître les paramètres de connexion (i.e. le nom d'hote, le nom d'utilisateur, et le mot de passe). Une fois que vous connaîtrez tous ces paramètres, vous pourrez vous connecter comme ceci :

```
shell62; mysql -h host -u user -p
Enter password: ********
```

Les ************************ représentent votre mot de passe : saisissez le lorsque ***mysql*** affiche the ***Enter password (Entrez votre mot de passe):*** invite.

Si tout a bien fonctionné, vous devriez voir s'afficher des informations d'introduction, suivies d'une invite de commande : ***mysql>*** prompt:

```
shell62; mysql -h host -u user -p
Enter password: ********
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 459 to server version: 3.22.20a-log

Type 'help' for help.
mysql62;
```

L'invite de commande vous indique que ***mysql*** est prêt à recevoir des commandes.

Certaines installations de ***MySQL*** permettent l'accès anonyme au serveur. Si c'est le cas de votre machine,

vous devriez pouvoir vous connecter sans fournir aucune information.

```
shell62; mysql
```

Après s'être correctement connecté, vous pouvez vous déconnecter à tout moment, en tapant ***QUIT*** à l'invite de ***mysql.***

```
mysql62; QUIT
Bye
```

Vous pouvez aussi vous déconnecter en tapant les touches contrôle−D.

Par la suite, nous supposerons que vous vous êtes correctement connecté au serveur.

8.2 Soumettre une requête

Assurez vous que vous êtes correctement connecté. Dans le cas contraire, reportez vous à la section précédente. Ce faisant, vous ne vous êtes en fait connecté à aucune base de données, mais c'est bien comme ça. A ce stade ; il est important de savoir comment envoyer une requête, avant de savoir créer une table, charger des données, et interroger la base. Cette section décrit les principes de base de saisie des commandes, en utilisant des commandes qui vous familiariseront avec le fonctionnement de ***MySQL.***

Voici une commande simple qui demande au serveur la version et la date courante. Saisissez la comme ci−dessous, puis appuyez sur la touche entrée.

```
mysql62; SELECT VERSION(), CURRENT_DATE;
+--------------+--------------+
| version()    | CURRENT_DATE |
+--------------+--------------+
| 3.22.20a-log | 1999-03-19   |
+--------------+--------------+
1 row in set (0.01 sec)
mysql62;
```

Cette première requête montre beaucoup de caractéristiques de ***mysql***

Une commande consiste généralement d'une commande SQL, suivie d'un point−virgule (Il y a quelques exceptions, ou les point−virgules ne sont pas nécessaires, comme la commande ***QUIT*** , vue précédement. Nous y reviendrons plus loin).

Quand une requête a été saisie, ***mysql*** l'envoie au serveur pour qu'il l'exécute, puis affiche le résultat, et repropose une nouvelle invite de commande : ***mysql>.***

Mysql> affiche la réponse du serveur sous forme de table (lignes et colonnes). La première ligne contient les titres des colonnes. Les lignes suivantes présentent les résultats de la requête. Généralement, les noms de colonnes sont les noms des colonnes des tables utilisées. Si la valeur retournée est une expression plutôt qu'une table, (comme dans l'exemple ci−dessus), ***mysql*** crée une colonne avec comme titre l'expression évaluée.

mysql affiche le nombre de ligne retourné, et le temps de traitement de la requête, ce qui donne une idée de la performance globale du serveur. Ces valeurs sont imprécises, car elle représente le temps passé entre l'envoi de la commande et la réception de la réponse, et ne montre pas quelle quantité de processeur a été

utilisée. Cela ne permet pas de connaître la charge du serveur, ou les retards du réseau.

Par un souci de concision, la ligne ``rows in set'' ne sera plus affichée dans les exemples ultérieurs.

Les mots clés du langage peuvent être en majuscule ou minuscule, au choix. Les lignes suivantes sont équivalentes :

```
mysql62; SELECT VERSION(), CURRENT_DATE;
mysql62; select version(), current_date;
mysql62; SeLeCt vErSiOn(), current_DATE;
```

Voici une autre requête qui montre que ***mysql*** peut être utilisé comme une simple calculatrice.

```
mysql62; SELECT SIN(PI()/4), (4+1)*5;
+------------+---------+
| SIN(PI()/4) | (4+1)*5 |
+------------+---------+
|    0.707107 |      25 |
+------------+---------+
```

Les commandes que nous venons de voir sont relativement courtes, et tiennent sur une seule ligne. Il est possible de saisir plusieurs commandes sur une seule ligne, il suffit de toujours les terminer par des points−virgules.

```
mysql62; SELECT VERSION(); SELECT NOW();
+--------------+
| version()    |
+--------------+
| 3.22.20a−log |
+--------------+

+---------------------+
| NOW()               |
+---------------------+
| 1999−03−19 00:15:33 |
+---------------------+
```

Une commande n'est pas obligatoirement sur une seule ligne : les commandes les plus longues peuvent tenir sur plusieurs lignes. Ce n'est pas un problème, car ***mysql*** détermines la fin de la commandes gr ce au point−virgule, et non pas en cherchant la fin de la ligne (en d'autres termes, ***mysql*** accepte n'importe quel format de colonne, mais ne les exécute que si il trouve un point−virgule à la fin de la commande).

Voici une commande simple, et multi−lignes :

```
mysql62; SELECT
    −62; USER()
    −62; ,
    −62; CURRENT_DATE;
+--------------------+--------------+
| USER()             | CURRENT_DATE |
+--------------------+--------------+
| joesmith@localhost | 1999−03−18   |
+--------------------+--------------+
```

Dans cet exemples, vous avez pu remarquer que l'invite passe de ***mysql>*** à **−>** dès que la commande devient multi−lignes. C'est par ce biais que ***mysql*** indique qu'il n'a pas trouvé une commande complète, et qu'il

attend un complément d'information. En observant bien l'invite de commande, vous saurez toujours ce que ***mysql*** attend de vous.

Pour annuler une commande qui est partiellement saisie, il suffit de taper '\c' (slash−c)

```
mysql62; SELECT
    −62; USER()
    −62; \c
mysql62;
```

Ici, l'invite de commande reprend son aspect initial. Cela indique que ***mysql*** est prêt pour une nouvelle commande.

La table suivante montre les différentes formes de l'invite de commande, et sa signification :

Une commande peut s'étendre sur plusieurs lignes si, par accident, vous oubliez de terminer votre ligne par un point−virgule. Dans ce cas, ***mysql*** attend plus d'informations :

```
mysql62; SELECT USER()
    −62;
```

Si cela vous arrive (vous pensez avoir entré une commande, mais la seule réponse est cette désespérante invite **−>**) ; le plus souvent ***mysql*** attends le point−virgule. Si vous ne comprenez pas que ***mysql*** attend la suite de votre commande, vous risquez d'attendre un bon moment. Il suffit alors de compléter la commande avec un point−virgule, pour valider la commande.

```
mysql62; SELECT USER()
    −62; ;
+−−−−−−−−−−−−−−−−−−−+
| USER()            |
+−−−−−−−−−−−−−−−−−−−+
| joesmith@localhost |
+−−−−−−−−−−−−−−−−−−−+
```

Les formes **'>** et **">** d'invite de commande apparaissent lors de la saisie de chaînes. Avec ***MySQL***, vous pouvez écrire des chaînes avec les guillemets simples et doubles : ``'''' ou ``"" ; 'comme par exemple ***'bonjour'*** ou ***"au revoir"***. **'>** et **">** signifie donc que vous avez commencé à saisir une chaîne de caractères, mais que vous n'avez pas encore fini. Si vous saisissez une chaîne de plusieurs lignes, c'est une indication judicieuse, mais est−ce souvent le cas? En général, ces deux invites de commande indiquent que vous avez oublié de refermer les guillemets :

```
mysql62; SELECT * FROM my_table WHERE nom = "Smith AND age 60; 30;
    "62;
```

Si vous saisissez cette commande ***SELECT*** , puis tapez ENTREE, il ne va rien se passer. Plutôt que de se demander " mais qu'est ce qui prend tant de temps ", il vaut mieux remarquer que l'invite a pris la forme particulière de **">** . Cela signifie que ***mysql*** s'attend ce que vous complétiez votre chaîne et la commande. En effet, la chaîne ***"Smith*** n'a pas de deuxième guillemet.

A ce moment, que faire ? La chose la plus simplet d'annuler la commande. Cependant, vous ne pouvez pas taper **\c** , car ***mysql*** l'interprétera comme un caractère de chaîne. A la place, il faut clore la chaîne, puis taper **\c** .

```
mysql62; SELECT * FROM my_table WHERE nom = "Smith AND age 60; 30;
```

```
    "62; "\c
mysql62;
```

L'invite de commande redevient ***mysql>***, ce qui indique que ***mysql*** est prêt pour une nouvelle commande.

Il est important que les invites de commande de la forme signifie ***'>*** et ***">*** que vous n'avez pas terminé une chaîne, et que toutes les lignes suivantes seront ignorées par l'analyseur de ***mysql*** – y compris la commande ***QUIT***! Cela peut être déroutant, surtout si vous ne savez pas qu'il faut absolument fournir un guillemet de fin, même pour annuler la saisie

8.3 Exemples de requêtes

Voici quelques exemples de requêtes classiques avec ***MySQL***.

Certains des exemples utilisent la table 'shop' qui contient les prix de chaque article (numéro d'objet). Supposons que chaque objet a un prix unique, et que le couple (item, trader) et une clé prmiaire pour ces lignes.

Vous pouvez créer cet exemple avec la table suivante :

```
CREATE TABLE shop (
 article INT(4) UNSIGNED ZEROFILL DEFAULT '0000' NOT NULL,
 dealer  CHAR(20)                 DEFAULT ''     NOT NULL,
 price   DOUBLE(16,2)             DEFAULT '0.00' NOT NULL,
 PRIMARY KEY(article, dealer));

INSERT INTO shop VALUES
(1,'A',3.45),(1,'B',3.99),(2,'A',10.99),(3,'B',1.45),(3,'C',1.69),
(3,'D',1.25),(4,'D',19.95);
```

Les données pour l'exemple sont :

```
SELECT * FROM shop

+---------+--------+-------+
| article | dealer | price |
+---------+--------+-------+
|    0001 | A      |  3.45 |
|    0001 | B      |  3.99 |
|    0002 | A      | 10.99 |
|    0003 | B      |  1.45 |
|    0003 | C      |  1.69 |
|    0003 | D      |  1.25 |
|    0004 | D      | 19.95 |
+---------+--------+-------+
```

8.3.1 Valeur maximale d'une colone

"Quel est le plus grand numéro d'objet?"

```
SELECT MAX(article) AS article FROM shop

+---------+
| article |
+---------+
```

```
|       4 |
+---------+
```

8.3.2 La ligne qui contient le maximum d'une colonne

"Retrouver le prix, le vendeur et le numéro de l'objet le plus cher du magasin"

En ANSI−SQL cela est très facilement fait avec un sous selection :

```
SELECT article, dealer, price
FROM   shop
WHERE  price=(SELECT MAX(price) FROM shop)
```

Avec ***MySQL*** (et donc, sans les sous selections), il faut le faire en deux étapes :

1. Retrouver la valeur maximale de la table, avec la commande `SELECT`.
2. Avec la valeur lue, créer la requêt suivante :
   ```
   SELECT article, dealer, price
   FROM   shop
   WHERE  price=19.95
   ```

Une autre solution est de trier les objets par prix, et de lire la première ligne, avec la clause ***MySQL*** `LIMIT`:

```
SELECT article, dealer, price
FROM   shop
ORDER BY price DESC
LIMIT 1
```

Note: Avec cette méthode, on ne verra qu'ne seul objet, même si il y a plusieurs objets de meme prix.

8.3.3 Maximum d'une colonne : par groupement

"Quel est le prix maximal d'un article?"

```
SELECT article, MAX(price) AS price
FROM   shop
GROUP BY article
```

```
+---------+-------+
| article | price |
+---------+-------+
|    0001 |  3.99 |
|    0002 | 10.99 |
|    0003 |  1.69 |
|    0004 | 19.95 |
+---------+-------+
```

8.3.4 La ligne contenant le maximum d'une colonne d'un groupe

"Pour chaque article, trouver le vendeur le plus cher."

En `ANSI SQL` on pourrai le faire avec une sous selection, comme ceci :

```
SELECT article, dealer, price
FROM   shop s1
```

```
WHERE  price=(SELECT MAX(s2.price)
              FROM shop s2
              WHERE s1.article = s2.article)
```

Avec ***MySQL*** il vaut mieux le faire en deux étapes :

1. Retrouver la liste des (article,prix_maxima). 8.3.4 La ligne contenant le maximum d'une colonne d'un groupe.
2. pour chaque article trouvé, retrouver la ligne correspondante, pour lire le nom du vendeur. price.

Cela peut se faire facilement avec une table temporaire:

```
CREATE TEMPORARY TABLE tmp (
        article INT(4) UNSIGNED ZEROFILL DEFAULT '0000' NOT NULL,
        price   DOUBLE(16,2)             DEFAULT '0.00' NOT NULL);

LOCK TABLES article read;

INSERT INTO tmp SELECT article, MAX(price) FROM shop GROUP BY article;

SELECT article, dealer, price FROM shop, tmp
WHERE shop.article=tmp.articel AND shop.price=tmp.price;

UNLOCK TABLES;

DROP TABLE tmp;
```

Si vous n'utislisez pas de table temporaire, il vous faut verrouiller la table.

"Est ce qu'il est impossible de faire cela avec une seule requête?"

Oui, mais en utilisant une astuce qui s'appelle : "MAX−CONCAT trick":

```
SELECT article,
       SUBSTRING( MAX( CONCAT(LPAD(price,6,'0'),dealer) ), 7) AS dealer,
  0.00+LEFT(      MAX( CONCAT(LPAD(price,6,'0'),dealer) ), 6) AS price
FROM   shop
GROUP BY article;

+---------+--------+-------+
| article | dealer | price |
+---------+--------+-------+
|    0001 | B      |  3.99 |
|    0002 | A      | 10.99 |
|    0003 | C      |  1.69 |
|    0004 | D      | 19.95 |
+---------+--------+-------+
```

Le dernier exemple peut être fait de manière plus efficace, en effectuant la scission de la colonne au niveau du client; The last example can of course be made a bit more efficient by doing the

8.3.5 Utiliser des clés étrangères

Il n'y a pas besoin de clé étrangère pour joindre deux tables.

La seule chose que `MySQL` ne fait pas est de `CHECK` (vérifier) que les clés que vous utilisez existent vraiment dans la table que vous réféencez, et qu'il n'efface par de lignes dnas une table avec une définition de clé étrangère. Si vous utilisez vos clés de manière habituelle, cela fonctionnera parfaitement.

```
CREATE TABLE persons (
    id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
    name CHAR(60) NOT NULL,
    PRIMARY KEY (id)
);

CREATE TABLE shirts (
    id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
    style ENUM('t−shirt', 'polo', 'dress') NOT NULL,
    color ENUM('red', 'blue', 'orange', 'white', 'black') NOT NULL,
    owner SMALLINT UNSIGNED NOT NULL REFERENCES persons,
    PRIMARY KEY (id)
);

INSERT INTO persons VALUES (NULL, 'Antonio Paz');

INSERT INTO shirts VALUES
(NULL, 'polo', 'blue', LAST_INSERT_ID()),
(NULL, 'dress', 'white', LAST_INSERT_ID()),
(NULL, 't−shirt', 'blue', LAST_INSERT_ID());

INSERT INTO persons VALUES (NULL, 'Lilliana Angelovska');

INSERT INTO shirts VALUES
(NULL, 'dress', 'orange', LAST_INSERT_ID()),
(NULL, 'polo', 'red', LAST_INSERT_ID()),
(NULL, 'dress', 'blue', LAST_INSERT_ID()),
(NULL, 't−shirt', 'white', LAST_INSERT_ID());

SELECT * FROM persons;
+----+--------------------+
| id | name               |
+----+--------------------+
|  1 | Antonio Paz        |
|  2 | Lilliana Angelovska |
+----+--------------------+

SELECT * FROM shirts;
+----+---------+--------+-------+
| id | style   | color  | owner |
+----+---------+--------+-------+
|  1 | polo    | blue   |     1 |
|  2 | dress   | white  |     1 |
|  3 | t−shirt | blue   |     1 |
|  4 | dress   | orange |     2 |
|  5 | polo    | red    |     2 |
|  6 | dress   | blue   |     2 |
|  7 | t−shirt | white  |     2 |
+----+---------+--------+-------+

SELECT s.* FROM persons p, shirts s
 WHERE p.name LIKE 'Lilliana%'
   AND s.owner = p.id
   AND s.color 60;> 'white';

+----+-------+--------+-------+
| id | style | color  | owner |
+----+-------+--------+-------+
|  4 | dress | orange |     2 |
|  5 | polo  | red    |     2 |
|  6 | dress | blue   |     2 |
+----+-------+--------+-------+
```

8.4 Créer et utiliser une base de données

Si l'administrateur vous a créé une base de données pour vous, alors vous pouvez directement commencer à l'utiliser. Sinon, il vous faut la créer vous même :

```
mysql62; CREATE DATABASE menagerie;
```

Sous Unix, les noms de base de données sont sensibles à la casse (contrairement aux mots clés SQL), donc il faudra faire référence à votre base de données sous le nom ***menagerie***, et non pas ***Menagerie***, ***MENAGERIE*** ou tout autre variante. Sous Windows, cette restriction ne s'applique pas, même si vous devez faire référence à vos bases et tables de la même manière tout au long d'une même commande).

Créer une base de données ne la selectionne pas automatiquement. Il faut le faire explicitement. Pour faire de ***menagerie*** votre base courante, il faut utiliser la commande:

```
mysql62; USE menagerie
Database changed
```

La base n'a besoin d'être crée qu'une seule fois, mais il faudra la sélectionner à chaque fois que vous commencerez une session ***mysql***. Il suffira alors d'utiliser la même commande que ci−dessus. Alternativement, vous pouvez sélectionner une base dès la connexion, en passant le nom de la base après tous les paramètres de connexion : .

```
shell62; mysql −h host −u user −p menagerie
Enter password: ********
```

Remarquez bien que ***menagerie*** n'est pas dans votre mot de passe. Si vous voulez transmettre votre mot de passe après l'option ***−p***, vous devez le faire sans espace entre le mot de passe et l'option : (e.g., tel que ***−pmypassword***, mais pas ***−p mypassword***). Cependant, mettre votre mot de passe dans la ligne de connexion n'est pas très recommandé, car cela vous rend vulnérable à tous les mouchards qui pourraient être sur votre machine.

8.4.1 Creéer une table

Créer une base de données est facile, mais, jusqu'à présent, c'est vide. La commande ***SHOW TABLES*** vous dira :

```
mysql62; SHOW TABLES;
Empty set (0.00 sec)
```

La partie la plus difficile est le choix de la structure de votre base de données, et des tables dont vous aurez besoin, et quelles colonnes seront nécessaires.

Vous pouvez envisager de créer une table qui créera un enregistrement pour chacun de vos animaux. Cette table portera le nom de ***animaux*** et devrait contenir au minimum le nom de l'animal. Etant donné que le nom seul n'est pas vraiment intéressant, il faudra qu'il contienne aussi d'autres informations. Par exemple, si plusieurs personnes de votre famille ont des animaux domestiques, vous voudrez garder la liste de chaque maître. Vous voudrez peut être aussi conserver des informations basiques telles que le genre ou la race.

Et l'age ? Cela pourrait être intéressant à conserver, mais ce n'est pas une bonne chose à conserver dans une base de données. En effet, l'age change tous les jours, et il faudrait changer constamment la base de données.

Au contraire, il est bien mieux de conserver la date de naissance. Alors, à chaque fois que vous aurez besoins de l'age, il suffira de faire la différence entre la date du jour et la date de naissance. ***MySQL*** disposent de puissantes fonctions de calculs sur les dates. Enregistrer la date de naissance plutôt quel l'age a d'autres atouts :

Vous pourrez utiliser la base de données pour garder en mémoire les dates d'anniversaires de vos animaux (Si cela vous semble un peu idiot, remarquez bien que c'est exactement la même chose que de conserver la date d'anniversaire de vos clients, et de leur envoyer cette carte d'anniversaire à la spontanéité toute informatique).

Vous pourrez faire des calculs d'age en relation avec d'autres dates. Par exemple, si vous enregistrer la date de mort, vous pourrez facilement calculer à quel age est mort votre compagnon.

Votre imagination fertile vous permettra sûrement d'imaginer une foule d'informations utiles pour garnir la table ***animaux*** , mais les champs que nous venons d'identifier seront suffisant pour l'instant : le nom, le propriétaire, la race, le genre, la date de naissance et celle de mort.

Utilisez maintenant la fonction de création de table pour créer la votre :

```
mysql62; CREATE TABLE animaux (nom VARCHAR(20), proprietaire VARCHAR(20),
    −62; espece VARCHAR(20), genre CHAR(1), naissance DATE, mort DATE);
```

VARCHAR est un bon choix pour le nom, le propriétaire et la race, car ces valeurs auront des longueurs variables. Les longueurs de ces colonnes n'ont pas besoin d'être toutes identiques, ni de valoir 20. Vous pouvez choisir n'importe quelle longueur entre 1 et 255, du moment que cela vous semble approprié (si vous vous trompez , vous pourrez toujours agrandir le champs avec la fonction ***MySQL*** : ***ALTER TABLE***).

Le genre des animaux peu prendre de nombreuses formes, comme par exemple ***"m"*** et ***"f"***, ou peut être ***"male"*** et ***"femelle"***. Le plus simple sera d'utiliser les caractères ***"m"*** et ***"f"***.

L'utilisation du type ***DATE*** pour représenter les dates de naissance ***naissance*** et de mort ***mort*** est un choix évident.

Maintenant que vous avez créer une table, , ***SHOW TABLES*** devrait être plus loquace :

```
mysql62; SHOW TABLES;
+---------------------+
| Tables in menagerie |
+---------------------+
| animaux             |
+---------------------+
```

Pour vérifier que la table a été créée comme vous le désiriez, utilisez la commande ***DESCRIBE*** :

```
mysql62; DESCRIBE animaux;
+---------------+-------------+------+-----+---------+-------+
| Field         | Type        | Null | Key | Default | Extra |
+---------------+-------------+------+-----+---------+-------+
| nom           | varchar(20) | YES  |     | NULL    |       |
| proprietaire  | varchar(20) | YES  |     | NULL    |       |
| espece        | varchar(20) | YES  |     | NULL    |       |
| genre         | char(1)     | YES  |     | NULL    |       |
| naissance     | date        | YES  |     | NULL    |       |
| mort          | date        | YES  |     | NULL    |       |
+---------------+-------------+------+-----+---------+-------+
```

Vous pouvez utiliser ***DESCRIBE*** à tout moment, par exemple, si vous oubliez les noms de colonnes ou leur type.

8.4.2 Charger des données dans une table

Après avoir créé votre table, il faut la remplir. La fonction ***LOAD DATA*** et ***INSERT*** remplissent cette fonction.

Supposons que les informations sur vos animaux soient décrites comme dans le tableau ci−dessous : Remaquez bien que ***MySQL*** utilise un format de date de type ***AAAA−MM−JJ*** ; qui n'est pas le format standard.)

Etant donné que vous commencez avec une table vide, le meilleur moyen de remplir cette table est de créer un fichier texte, chaque ligne contenant les informations d'un animal, puis de le charger directement dans la table avec une seule commande.

Vous créez ainsi un fichier ***animaux.txt'*** contenant un enregistrement par ligne, avec des valeurs séparées par des tabulation, et dans le même ordre que l'ordre dans lequel les colonnes ont été listées dans la commande ***CREATE TABLE***. Pour les valeurs manquantes (comme par exemple, les genres inconnues, ou les dates de mort des animaux vivants), vous pouvez utiliser la valeur ***NULL*** . Vous la représenterez dans le texte avec ***\N***. Par exemple, l'enregistrement de l'oiseau Whistler ressemblera à ceci :

Pour charger ce fichier ***`animaux.txt'*** dans la table ***animaux*** , utilisez la commande suivante :

```
mysql62; LOAD DATA LOCAL INFILE "animaux.txt" INTO TABLE animaux;
```

Vous pourriez spécifier le type de chaque colonne et le marqueur de fin de ligne dans la commande ***LOAD DATA*** si vous le désiriez, mais les valeurs par défaut (tabulations et retour chariot) fonctionnent très bien ici.

Pour n'ajouter qu'un seul enregistrement à la fois, la fonction ***INSERT*** est plus pratique : Dans sa forme la plus simple, vous fournissez les valeurs dans l'ordre des colonnes. Par exemple, si Diane recoit un hamster du nom de Puffball, vous pourriez ajouter un nouvel enregistrement avec la commande suivante :

```
mysql62; INSERT INTO animaux
    −62; VALUES ('Puffball','Diane','hamster','f','1999−03−30',NULL);
```

Notez bien que les chaînes et les dates sont spécifiées avec des guillemets. De la même façon, vous pouvez insérer la valeur ***NULL*** directement pour représenter une valeur manquante. N'utilisez pas ***\N*** comme pour ***LOAD DATA***.

A partir de cet exemple, vous voyez que la commande ***INSERT*** requiert nettement plus de frappe au clavier que la fonction ***LOAD DATA***

8.4.3 Lire des informations dans une table

La commande SELECT sert à lire des informations d'une table. La forme générale est la suivante :

```
SELECT what_to_select
FROM which_table
WHERE conditions_to_satisfy
```

`what_to_select` indique ce que vous voulez affichier. Cela peut être une liste de champs, ou bien le joker * qui signifie ``toutes les colonnes'' `which_table` indique dans quelle table lire les informations. La clause `WHERE` est optionnelle. Si elle est présente, `conditions_to_satisfy` spécifie les conditions qu'une ligne doit remplir pour être retenue, et retournée.

8.4.3.1 Selection toutes les données

La forme la plus simple de ***SELECT*** permet d'obtenir la liste complète des ligne d'une table :

```
mysql62; SELECT * FROM animaux;
+----------+--------------+---------+--------+----------------+------------+
| nom      | proprietaire | espece  | genre  | naissance      | mort       |
+----------+--------------+---------+--------+----------------+------------+
| Fluffy   | Harold       | chat    | f      | 1993-02-04     | NULL       |
| Claws    | Gwen         | chat    | m      | 1994-03-17     | NULL       |
| Buffy    | Harold       | chien   | f      | 1989-05-13     | NULL       |
| Fang     | Benny        | chien   | m      | 1990-08-27     | NULL       |
| Bowser   | Diane        | chien   | m      | 1998-08-31     | 1995-07-29 |
| Chirpy   | Gwen         | oiseau  | f      | 1998-09-11     | NULL       |
| Whistler | Gwen         | oiseau  | NULL   | 1997-12-09     | NULL       |
| Slim     | Benny        | serpent | m      | 1996-04-29     | NULL       |
| Puffball | Diane        | hamster | f      | 1999-03-30     | NULL       |
+----------+--------------+---------+--------+----------------+------------+
```

Cette forme de ***SELECT*** est utile pour passer en revue une table, comme par exemple, une table que vous viendriez de charger. Dans cet exemple, la table ci−dessus montre qu'il y a eu une erreur dans le fichier. Bowser semble être né après être mort ! En consultant son dossier, vous vous apercevez que sa date correcte de naissance est 1989, et non pas 1998.

Il y a au moins deux façons de corriger cette erreur :

Editez le fichier ***`animaux.txt'*** pour corriger l'erreur, puis effacer la table ,et la recharger avec la ***DELETE*** et ***LOAD DATA***:

```
mysql62; DELETE FROM animaux;
mysql62; LOAD DATA LOCAL INFILE "animaux.txt" INTO TABLE animaux;
```

Cependant, en faisant cela, il vous faudra aussi insérer de nouveau la fiche de Puffball.

Ou bien, corriger seulement la fiche erronée avec une commande ***UPDATE*** :

```
mysql62; UPDATE animaux SET naissance = "1989−08−31" WHERE nom = "Bowser";
```

Dans cet exemple, on voit qu'il est facile de sélectionner toute la table. Mais généralement, ce n'est pas très pratique, surtout quand la table devient trop grande. En général, il s'agit de réponse à une question plus spécifique, pour laquelle il va falloir ajouter des contraintes sur les informations à retourner. Voyons maintenant quelques exemples de requêtes.

8.4.3.2 Selectioner une partie des lignes

Il est bien sûr possible de ne sélectionner quelques lignes dans une table. Mettons que vous souhaitiez vérifier que la nouvelle date de naissance de Bowser's a bien été prise en compte. Il suffit de sélectionner l'enregistrement de Bowser comme ceci :

```
mysql62; SELECT * FROM animaux WHERE nom = "Bowser";
+--------+-------------+---------+--------+---------------+------------+
| nom    | proprietaire | espece  | genre  | naissance      | mort       |
+--------+-------------+---------+--------+---------------+------------+
| Bowser | Diane        | chien   | m      | 1989−08−31     | 1995−07−29 |
+--------+-------------+---------+--------+---------------+------------+
```

Le résultat confirme bien que l'année de naissance est 1989, et non plus 1998.

Les comparaisons de chaîne sont généralement insensible à la casse : on aurait plus préciser le nom `"bowser"`, `"BOWSER"`, etc. Le résultat aurait été le même.

Vous pouvez faire des recherches sur d'autres colonnes que `nom`. Par exemple, si vous voulez savoir quels animaux sont nés 1998, faites un test sur la colonne `naissance` :

```
mysql62; SELECT * FROM animaux WHERE naissance 62;= "1998−1−1";
+----------+-------------+---------+--------+---------------+------------+
| nom      | proprietaire | espece  | genre  | naissance      | mort       |
+----------+-------------+---------+--------+---------------+------------+
| Chirpy   | Gwen         | oiseau  | f      | 1998−09−11     | NULL       |
| Puffball | Diane        | hamster | f      | 1999−03−30     | NULL       |
+----------+-------------+---------+--------+---------------+------------+
```

Vous pouvez aussi combiner les conditions : par exemple, pour rechercher les chiennes

```
mysql62; SELECT * FROM animaux WHERE espece = "chien" AND genre = "f";
+----------+-------------+---------+--------+---------------+------------+
| nom      | proprietaire | espece  | genre  | naissance      | mort       |
+----------+-------------+---------+--------+---------------+------------+
| Buffy    | Harold       | chien   | f      | 1989−05−13     | NULL       |
+----------+-------------+---------+--------+---------------+------------+
```

La requête précédente utilisait l'opérateur logique `AND` (ET) Il y a aussi un opérateur `OR` (OU) :

```
mysql62; SELECT * FROM animaux WHERE espece = "serpent" OR espece = "oiseau";
+----------+-------------+---------+--------+---------------+------------+
| nom      | proprietaire | espece | genre  | naissance      | mort       |
+----------+-------------+---------+--------+---------------+------------+
| Chirpy   | Gwen         | oiseau  | f      | 1998−09−11     | NULL       |
| Whistler | Gwen         | oiseau  | NULL   | 1997−12−09     | NULL       |
| Slim     | Benny        | serpent | m      | 1996−04−29     | NULL       |
+----------+-------------+---------+--------+---------------+------------+
```

`AND` et `OR` peut être utilisés dans la même requête. C'est alors une bonne idée d'utiliser des parenthèses pour préciser les regroupements :

```
mysql62; SELECT * FROM animaux WHERE (espece = "chat" AND genre = "m")
    −62; OR (espece = "chien" AND genre = "f");
+----------+-------------+---------+--------+---------------+------------+
| nom      | proprietaire | espece  | genre  | naissance      | mort       |
+----------+-------------+---------+--------+---------------+------------+
| Claws    | Gwen         | chat    | m      | 1994−03−17     | NULL       |
| Buffy    | Harold       | chien   | f      | 1989−05−13     | NULL       |
+----------+-------------+---------+--------+---------------+------------+
```

8.4.3.3 Selectionner une colonne spécifique

Il se peut que vous n'ayez pas besoin de toutes les colonnes de votre table, mais juste de quelques colonnes. Il suffit alors de citer les colonnes qui vous intéressent. Par exemple, si vous ne voulez voir que les noms des animaux, avec leur date de naissance, il suffit de ne sélectionner que les colonnes `nom` et `naissance`:

```
mysql62; SELECT nom, naissance FROM animaux;
+----------+------------+
| nom      | naissance  |
+----------+------------+
| Fluffy   | 1993−02−04 |
| Claws    | 1994−03−17 |
| Buffy    | 1989−05−13 |
| Fang     | 1990−08−27 |
| Bowser   | 1989−08−31 |
| Chirpy   | 1998−09−11 |
| Whistler | 1997−12−09 |
| Slim     | 1996−04−29 |
| Puffball | 1999−03−30 |
+----------+------------+
```

Pour lister les propriétaires d'animaux, utilisez la requête suivante :

```
mysql62; SELECT proprietaire FROM animaux;
+---------------+
| proprietaire  |
+---------------+
| Harold        |
| Gwen          |
| Harold        |
| Benny         |
| Diane         |
| Gwen          |
| Gwen          |
| Benny         |
| Diane         |
+---------------+
```

Cependant, vous pouvez remarquer que cette requête simple affiche le champs `proprietaire` de chaque ligne, ce qui conduit à avoir des redondances (comme Gwen). Pour ne les voir apparaître qu'une seule fois, il faut utiliser le mot clé `DISTINCT`:

```
mysql62; SELECT DISTINCT proprietaire FROM animaux;
+---------------+
| proprietaire  |
+---------------+
| Benny         |
| Diane         |
| Gwen          |
| Harold        |
+---------------+
```

Vous pouvez encore combiner une clause `WHERE` lors de la selection de lignes et de colonnes Par exemple, pour obtenir les dates de naissances des chiens et des chats, utilisez la requête suivante :

```
mysql62; SELECT nom, espece, naissance FROM animaux
    −62; WHERE espece = "chien" OR espece = "chat";
+--------+---------+------------+
```

```
| nom    | espece  | naissance  |
+--------+---------+------------+
| Fluffy | chat    | 1993-02-04 |
| Claws  | chat    | 1994-03-17 |
| Buffy  | chien   | 1989-05-13 |
| Fang   | chien   | 1990-08-27 |
| Bowser | chien   | 1989-08-31 |
+--------+---------+------------+
```

8.4.3.4 Trier les lignes

Vous avez pu remarquer que les lignes précédentes ont été affichées dans un ordre aléatoire. Comme il est plus facile d'analyser une requête dont les lignes ont été triées, il vaut mieux trier ces lignes avec la clause : `ORDER BY` :

Voici la liste des dates de naissances des animaux, classées par date :

```
mysql62; SELECT nom, naissance FROM animaux ORDER BY naissance;
+----------+------------+
| nom      | naissance  |
+----------+------------+
| Buffy    | 1989-05-13 |
| Bowser   | 1989-08-31 |
| Fang     | 1990-08-27 |
| Fluffy   | 1993-02-04 |
| Claws    | 1994-03-17 |
| Slim     | 1996-04-29 |
| Whistler | 1997-12-09 |
| Chirpy   | 1998-09-11 |
| Puffball | 1999-03-30 |
+----------+------------+
```

Pour inverser l'ordre de tri, ajoutez le mot clé `DESC` (descendant) après le nom de la colonne que vous classez.

```
mysql62; SELECT nom, naissance FROM animaux ORDER BY naissance DESC;
+----------+------------+
| nom      | naissance  |
+----------+------------+
| Puffball | 1999-03-30 |
| Chirpy   | 1998-09-11 |
| Whistler | 1997-12-09 |
| Slim     | 1996-04-29 |
| Claws    | 1994-03-17 |
| Fluffy   | 1993-02-04 |
| Fang     | 1990-08-27 |
| Bowser   | 1989-08-31 |
| Buffy    | 1989-05-13 |
+----------+------------+
```

Vous pouvez faire des classements avec plusieurs critères de tri. Par exemple, pour trier les animaux pas espèce, puis par naissance pour chaque type d'animaux, utilisez la requête suivante :

```
mysql62; SELECT nom, espece, naissance FROM animaux ORDER BY espece, naissance DESC;
+----------+---------+------------+
| nom      | espece  | naissance  |
+----------+---------+------------+
| Chirpy   | oiseau  | 1998-09-11 |
```

```
| Whistler | oiseau  | 1997−12−09 |
| Claws    | chat    | 1994−03−17 |
| Fluffy   | chat    | 1993−02−04 |
| Fang     | chien   | 1990−08−27 |
| Bowser   | chien   | 1989−08−31 |
| Buffy    | chien   | 1989−05−13 |
| Puffball | hamster | 1999−03−30 |
| Slim     | serpent | 1996−04−29 |
+----------+---------+------------+
```

Notez bien que le mot clé `DESC` ne s'applique qu'à la colonne le précédent immédiatement (`naissance`); `espece` étant trié dans l'ordre ascendant.

8.4.3.5 Calculs sur les dates

MySQL possède de puissantes fonctions pour effectuer des calculs sur les dates, comme par exemple, calculer un age, ou extraire des parties de date.

Pour déterminer l'age de chacun des animaux, il faut calculer la différence entre la naissance et la date courante. Puis, convertir ces deux dates en jours, et diviser le tout par 365, pour avoir le nombre d'année.

```
mysql62; SELECT nom, (TO_DAYS(NOW())−TO_DAYS(naissance))/365 FROM animaux;
+----------+-----------------------------------------+
| nom      | (TO_DAYS(NOW())−TO_DAYS(naissance))/365 |
+----------+-----------------------------------------+
| Fluffy   |                                    6.15 |
| Claws    |                                    5.04 |
| Buffy    |                                    9.88 |
| Fang     |                                    8.59 |
| Bowser   |                                    9.58 |
| Chirpy   |                                    0.55 |
| Whistler |                                    1.30 |
| Slim     |                                    2.92 |
| Puffball |                                    0.00 |
+----------+-----------------------------------------+
```

Bien que cette requête fasse bien ce qu'on lui demande, il y a de la place pour quelques améliorations. En premier lieu, les résultats gagneraient à être classés. De plus, le titre de la colonne n'est pas très explicite.

Le premier problème peut être résolu avec une clause `ORDER BY nom` qui va classer par ordre alphabétique. Pour régler le problème du titre, nous allons utiliser un alias.

```
mysql62; SELECT nom, (TO_DAYS(NOW())−TO_DAYS(naissance))/365 AS age
    −62; FROM animaux ORDER BY nom;
+----------+------+
| nom      | age  |
+----------+------+
| Bowser   | 9.58 |
| Buffy    | 9.88 |
| Chirpy   | 0.55 |
| Claws    | 5.04 |
| Fang     | 8.59 |
| Fluffy   | 6.15 |
| Puffball | 0.00 |
| Slim     | 2.92 |
| Whistler | 1.30 |
+----------+------+
```

Pour trier les résultats par `age` plutôt que par nom `nom`, il suffit de le mettre dans la clause `ORDER BY` :

```
mysql62;  SELECT nom, (TO_DAYS(NOW())−TO_DAYS(naissance))/365 AS age
    −62;  FROM animaux ORDER BY age;
+----------+------+
| nom      | age  |
+----------+------+
| Puffball | 0.00 |
| Chirpy   | 0.55 |
| Whistler | 1.30 |
| Slim     | 2.92 |
| Claws    | 5.04 |
| Fluffy   | 6.15 |
| Fang     | 8.59 |
| Bowser   | 9.58 |
| Buffy    | 9.88 |
+----------+------+
```

Une requête similaire pourrait calculer l'age de mort des animaux morts. Pour cela, vous allez déterminer les animaux morts, en testant la colonne `mort` à `NULL`. Puis, pour les valeurs non−`NULL`, calculez l'age avec les colonnes `mort` et `naissance`:

```
mysql62;  SELECT nom, naissance, mort, (TO_DAYS(mort)−TO_DAYS(naissance))/365 AS age
    −62;  FROM animaux WHERE mort IS NOT NULL ORDER BY age;
+--------+------------+------------+------+
| nom    | naissance  | mort       | age  |
+--------+------------+------------+------+
| Bowser | 1989−08−31 | 1995−07−29 | 5.91 |
+--------+------------+------------+------+
```

La requête utilise `mort IS NOT NULL` plutôt que `mort != NULL` car `NULL` est une valeur spéciale. Cela est expliqué plus loin. Allez 8.4.3.6 Travailler avec la valeur NULL.

Et comment rechercher les animaux dont l'anniversaire sera le mois prochain ? Pour ce genre de calculs, year et day sont inutiles, il suffit d'extraire le mois de la colonne `naissance` . ***MySQL*** fournit plusieurs fonctions d'extraction comme par exemple `YEAR()`, `MONTH()` et `DAY()`. `MONTH()` est le plus approprié ici. Pour voir comment cela fonction, exécutez la commande suivante, qui `naissance` et `MONTH(naissance)`:

```
mysql62; SELECT nom, naissance, MONTH(naissance) FROM animaux;
+----------+------------+-------------------+
| nom      | naissance  | MONTH(naissance)  |
+----------+------------+-------------------+
| Fluffy   | 1993−02−04 |                 2 |
| Claws    | 1994−03−17 |                 3 |
| Buffy    | 1989−05−13 |                 5 |
| Fang     | 1990−08−27 |                 8 |
| Bowser   | 1989−08−31 |                 8 |
| Chirpy   | 1998−09−11 |                 9 |
| Whistler | 1997−12−09 |                12 |
| Slim     | 1996−04−29 |                 4 |
| Puffball | 1999−03−30 |                 3 |
+----------+------------+-------------------+
```

Trouver les animaux dont la date de naissance est le mois prochain est facile. En supposant que nous soyons au mois d'avril. Alors, le mois est le 4, et il suffit de rechercher les animaux nés au mois de May (5), comme ceci :

```
mysql62; SELECT nom, naissance FROM animaux WHERE MONTH(naissance) = 5;
+-------+----------------+
| nom   | naissance      |
+-------+----------------+
| Buffy | 1989−05−13     |
+-------+----------------+
```

Il y a bien sur un cas particulier: décembre. Il ne suffit pas seulement d'ajouter 1 à numéro du mois courant et de chercher les dates d'anniversaires correspondantes, car personne ne nait au mois 13. A la place, il faut chercher les animaux qui sont nés au mois de janvier.

Vous pourriez écrire une requête qui fonctionne, quelque soit le mois courant. De cette façon, vous n'aurez pas à utiliser un numéro particulier de mois dans la requête. `DATE_ADD()` vous permettra d'ajouter une durée de temps à une date. Si vous ajoutez un mois à la date de `NOW()`, puis vous en sortez le mois avec `MONTH()`, le résultat sera bien le mois suivant.

```
mysql62; SELECT nom, naissance FROM animaux
    −62; WHERE MONTH(naissance) = MONTH(DATE_ADD(NOW(), INTERVAL 1 MONTH));
```

Une autre manière de faire serait d'ajouter `1` au mois courant, puis d'utiliser la (`MOD`) pour ``boucler'' à la fin de l'année, et faire correspondre janvier et décembre :

```
mysql62; SELECT nom, naissance FROM animaux
    −62; WHERE MONTH(naissance) = MOD(MONTH(NOW()),12) + 1;
```

8.4.3.6 Travailler avec la valeur `NULL`

La valeur `NULL` peut se comporter de manière surprenante si vous l'utilisez. Conceptuellement, `NULL` signifie ``valeur manquante '' ou `` valeur inconnue'' et il est traité de manière légèrement différente des autres valeurs. Pour tester une valeur à `NULL`, vous ne pouvez pas utiliser les opérateurs de comparaison habituels, tels que =, < or `!=`. Pour vous en convaincre, essayez la requête suivante :

```
mysql62; SELECT 1 = NULL, 1 != NULL, 1 60; NULL, 1 62; NULL;
+----------+----------+----------+----------+
| 1 = NULL | 1 != NULL | 1 60; NULL | 1 62; NULL |
+----------+----------+----------+----------+
|     NULL |      NULL |     NULL |     NULL |
+----------+----------+----------+----------+
```

Clairement, vous n'obtiendrez aucun résultat significatif de ces comparaisons. Utilisez les opérateurs `IS NULL` et `IS NOT NULL`:

```
mysql62; SELECT 1 IS NULL, 1 IS NOT NULL;
+-----------+---------------+
| 1 IS NULL | 1 IS NOT NULL |
+-----------+---------------+
|         0 |             1 |
+-----------+---------------+
```

Avec ***MySQL***, 0 signifie faux et 1 signifie vrai.

Cette gestion spéciale de `NULL` explique pourquoi, dans la section précédente, il était nécessaire de savoir quels animaux étaient encore vivant, en utilisant `mort IS NOT NULL` à la place de `mort != NULL`.

8.4.3.7 Recherche de valeurs

MySQL propose les méthodes de recherche standard du SQL, mais aussi les recherches à base d'expression régulière, similaire à celle utilisées dans les utilitaires Unix, tels que `vi`, `grep` et `sed`.

Les méthodes de recherche SQL vous permettent d'utiliser le caractère ``_'' pour remplacer n'importe quel caractère unique, et ``%'' pour remplacer n'importe quel nombre de caractères (y compris le caractère 0). Les recherches SQL sont insensibles à la casse. Reportez vous aux exemples ci−dessous. Remarquez bien que l'on n'utilise pas = ou != mais plutôt `LIKE` ou `NOT LIKE`.

Recherche des noms commençant par ``b'':

```
mysql62; SELECT * FROM animaux WHERE nom LIKE "b%";
+--------+--------------+--------+--------+---------------+------------+
| nom    | proprietaire | espece | genre  | naissance     | mort       |
+--------+--------------+--------+--------+---------------+------------+
| Buffy  | Harold       | chien  | f      | 1989−05−13    | NULL       |
| Bowser | Diane        | chien  | m      | 1989−08−31    | 1995−07−29 |
+--------+--------------+--------+--------+---------------+------------+
```

Recherche des noms finissant par :``fy'':

```
mysql62; SELECT * FROM animaux WHERE nom LIKE "%fy";
+--------+--------------+--------+--------+---------------+------------+
| nom    | proprietaire | espece | genre  | naissance     | mort       |
+--------+--------------+--------+--------+---------------+------------+
| Fluffy | Harold       | chat   | f      | 1993−02−04    | NULL       |
| Buffy  | Harold       | chien  | f      | 1989−05−13    | NULL       |
+--------+--------------+--------+--------+---------------+------------+
```

Recherche des noms contenant ``w'':

```
mysql62; SELECT * FROM animaux WHERE nom LIKE "%w%";
+----------+--------------+--------+--------+---------------+------------+
| nom      | proprietaire | espece | genre  | naissance     | mort       |
+----------+--------------+--------+--------+---------------+------------+
| Claws    | Gwen         | chat   | m      | 1994−03−17    | NULL       |
| Bowser   | Diane        | chien  | m      | 1989−08−31    | 1995−07−29 |
| Whistler | Gwen         | oiseau | NULL   | 1997−12−09    | NULL       |
+----------+--------------+--------+--------+---------------+------------+
```

Recherche des noms contenant exactement 5 caractères, utilisez le caractère ``_'' :

```
mysql62; SELECT * FROM animaux WHERE nom LIKE "_____";
+----------+--------------+--------+--------+---------------+------------+
| nom      | proprietaire | espece | genre  | naissance     | mort       |
+----------+--------------+--------+--------+---------------+------------+
| Claws    | Gwen         | chat   | m      | 1994−03−17    | NULL       |
| Buffy    | Harold       | chien  | f      | 1989−05−13    | NULL       |
+----------+--------------+--------+--------+---------------+------------+
```

L'autre type de recherche disponible avec ***MySQL*** est les expression régulières. Pour utiliser ce type de recherche, il faut ajouter les mots clé `REGEXP` et `NOT REGEXP` (ou `RLIKE` t `NOT RLIKE`, qui sont des synonymes).

Les caractéristiques des expressions régulières sont :

- ``.'' remplace n'importe quel caractère qui n'apparaît qu'une fois.
- Une classe de caractères ``[...]'' remplace n'importe quel caractère qui apparaît dans les crochets. Par exemple, ``[abc]'' peut remplacer ``a'', ``b'' ou ``c''. Pour un intervalle de caractères, utilisez le tiret :. ``[a−z]'' remplace n'importe quelle lettre minuscule, et ``[0−9]'' remplace n'importe quel nombre.
- ``*'' remplace zéro ou plus occurrences du caractère le précédent immédiatement. Par exemple, ``x*'' remplace n'importe quelle nombre de ``x''. ``[0−9]*'' `'' remplace n'importe quelle nombre de chiffres, et ``.*'' remplace n'importe quelle nombre de caractères.
- Les expression régulières sont sensibles à la casse, mais vous pouvez utiliser une classe de caractères pour les rendre insensible à la casse. Par exemple, ``[aA]'' remplace n'importe quel ``a'', minuscule ou majuscule, et ``[a−zA−Z]'' remplace n'importe quelle lettre, minuscule ou majuscule.
- La recherche est positive, si elle est vérifiée à n'importe quel endroit de la valeur (en SQL, ce n'est vrai que sur la valeur entière).
- Pour contraindre une expression au début ou à la fin de la valeur, utilisez les caractères spéciaux ``^'' pour le début ou ``$'' pour la fin.

Pour illustrer le fonctionnement des expressions régulières, les requêtes précédentes ont été réécrites en utilisant les expressions régulières.

Recherche des noms commençant par ``b'': on utilise ``^'' pour indiquer le début de la valeur, et ``[bB]'' pour rechercher indifféremment, ``b'' minuscule ou majuscule.

```
mysql62; SELECT * FROM animaux WHERE nom REGEXP "^[bB]";
+--------+---------------+--------+--------+----------------+------------+
| nom    | proprietaire  | espece | genre  | naissance      | mort       |
+--------+---------------+--------+--------+----------------+------------+
| Buffy  | Harold        | chien  | f      | 1989−05−13     | NULL       |
| Bowser | Diane         | chien  | m      | 1989−08−31     | 1995−07−29 |
+--------+---------------+--------+--------+----------------+------------+
```

Recherche des noms finissant par :``fy'': on utilise ``$'' pour indiquer la fin de la valeur

```
+--------+---------------+--------+--------+----------------+------------+
| nom    | proprietaire  | espece | genre  | naissance      | mort       |
+--------+---------------+--------+--------+----------------+------------+
| Fluffy | Harold        | chat   | f      | 1993−02−04     | NULL       |
| Buffy  | Harold        | chien  | f      | 1989−05−13     | NULL       |
+--------+---------------+--------+--------+----------------+------------+
```

Recherche des noms contenant ``w'':, on utilise ``[wW]'' pour rechercher les ``w', 'minuscule ou majuscule :

```
mysql62; SELECT * FROM animaux WHERE nom REGEXP "[wW]";
+----------+---------------+--------+--------+----------------+------------+
| nom      | proprietaire  | espece | genre  | naissance      | mort       |
+----------+---------------+--------+--------+----------------+------------+
| Claws    | Gwen          | chat   | m      | 1994−03−17     | NULL       |
| Bowser   | Diane         | chien  | m      | 1989−08−31     | 1995−07−29 |
| Whistler | Gwen          | oiseau | NULL   | 1997−12−09     | NULL       |
+----------+---------------+--------+--------+----------------+------------+
```

Etant donné qu'une expression régulière est vrai si elle est vrai sur une partie d'une valeur, il n'est pas besoin de caractères spéciaux.

Recherche des noms contenant exactement 5 caractères, utilisez ``^'' et ``$'' pour indiquer le début et la fin de la chaîne, et 5 fois ``.'' pour les 5 caractères.

```
mysql62; SELECT * FROM animaux WHERE nom REGEXP "^.....$";
+----------+---------------+--------+--------+----------------+------------+
| nom      | proprietaire  | espece | genre  | naissance      | mort       |
+----------+---------------+--------+--------+----------------+------------+
```

```
| Claws    | Gwen          | chat   | m      | 1994−03−17     | NULL       |
| Buffy    | Harold        | chien  | f      | 1989−05−13     | NULL       |
+----------+---------------+--------+--------+----------------+------------+
```

Vous auriez pu aussi utiliser l'opérateur ``{n''} `` n−fois":

```
mysql62; SELECT * FROM animaux WHERE nom REGEXP "^.{5}$";
+----------+---------------+--------+--------+----------------+------------+
| nom      | proprietaire  | espece | genre  | naissance      | mort       |
+----------+---------------+--------+--------+----------------+------------+
| Claws    | Gwen          | chat   | m      | 1994−03−17     | NULL       |
| Buffy    | Harold        | chien  | f      | 1989−05−13     | NULL       |
+----------+---------------+--------+--------+----------------+------------+
```

8.4.3.8 Compter les lignes

Les bases de données sont souvent utilisées pour répondre aux questions du type : ``combien de fois une information est−elle enregistrée dans une table?". Par exemple, vous pouvez souhaiter connaître le nombre d'animaux que vous avez, ou le nombre d'animaux de chaque propriétaire, ou encore toutes sortes de statistiques sur les animaux.

Pour compter le nombre total d'animaux que vous avez, il suffit de compter le nombre de ligne dans la table `animaux` , puisqu'il y a un enregistrement par animal. La fonction `COUNT()` compte le nombre de ligne non−`NULL`. Votre requête ressemblera alors à :

```
mysql62; SELECT COUNT(*) FROM animaux;
+----------+
| COUNT(*) |
+----------+
|        9 |
+----------+
```

Précédemment, vous avez recherché les noms des propriétaires d'animaux. Vous pouvez utiliser la fonction `COUNT()` pour connaître le nombre d'animaux que chaque propriétaire a :

```
mysql62; SELECT proprietaire, COUNT(*) FROM animaux GROUP BY proprietaire;
+---------------+----------+
| proprietaire  | COUNT(*) |
+---------------+----------+
| Benny         |        2 |
| Diane         |        2 |
| Gwen          |        3 |
| Harold        |        2 |
+---------------+----------+
```

Remarques: l'utilisation de la clause `GROUP BY` qui rassemble les lignes par `proprietaire`. Sans cette clause, vous obtenez le message d'erreur suivant :

```
mysql62; SELECT proprietaire, COUNT(proprietaire) FROM animaux;
ERROR 1140 at line 1: Mixing of GROUP columns (MIN(),MAX(),COUNT()...)
with no GROUP columns is illegal if there is no GROUP BY clause
```

`COUNT()` et `GROUP BY` sont utiles pour caractériser vos informations dans de nombreuses situations : Les exemples suivant effectuent des statistiques sur vos animaux :

Nombre d'animaux par espèce

```
mysql62; SELECT espece, COUNT(*) FROM animaux GROUP BY espece;
+----------+----------+
| espece   | COUNT(*) |
+----------+----------+
| oiseau   |        2 |
| chat     |        2 |
| chien    |        3 |
| hamster  |        1 |
| serpent  |        1 |
+----------+----------+
```

Nombre d'animaux par genre:

```
mysql62; SELECT genre, COUNT(*) FROM animaux GROUP BY genre;
+-------+----------+
| genre | COUNT(*) |
+-------+----------+
| NULL  |        1 |
| f     |        4 |
| m     |        4 |
+-------+----------+
```

(Dans cette réponse, `NULL` indique ``genre inconnu.'')

Nombre d'animaux par espece et genre:

```
mysql62; SELECT espece, genre, COUNT(*) FROM animaux GROUP BY espece, genre;
+---------+-------+----------+
| espece  | genre | COUNT(*) |
+---------+-------+----------+
| oiseau  | NULL  |        1 |
| oiseau  | f     |        1 |
| chat    | f     |        1 |
| chat    | m     |        1 |
| chien   | f     |        1 |
| chien   | m     |        2 |
| hamster | f     |        1 |
| serpent | m     |        1 |
+---------+-------+----------+
```

Il n'y a pas besoin d'utiliser la table entière avec la fonction `COUNT()`. Par exemple, la requête précédente effectuée sur la population de chien et de chat devient:

```
mysql62; SELECT espece, genre, COUNT(*) FROM animaux
    −62; WHERE espece = "chien" OR espece = "chat"
    −62; GROUP BY espece, genre;
+---------+-------+----------+
| espece  | genre | COUNT(*) |
+---------+-------+----------+
| chat    | f     |        1 |
| chat    | m     |        1 |
| chien   | f     |        1 |
| chien   | m     |        2 |
+---------+-------+----------+
```

Ou, pour avoir le nombre d'animaux par genre, et pour les espèces connues :

```
mysql62; SELECT espece, genre, COUNT(*) FROM animaux
    −62; WHERE genre IS NOT NULL
```

```
    −62; GROUP BY espece, genre;
+---------+-------+----------+
| espece  | genre | COUNT(*) |
+---------+-------+----------+
| oiseau  | f     |        1 |
| chat    | f     |        1 |
| chat    | m     |        1 |
| chien   | f     |        1 |
| chien   | m     |        2 |
| hamster | f     |        1 |
| serpent | m     |        1 |
+---------+-------+----------+
```

8.4.4 Utiliser plus d'une table

La table `animaux` contient la liste des animaux que vous avez. Vous pourriez vouloir enregistrer d'autres informations à leur sujet, telles que des évènements de leur vie, comme les visites chez le vétérinaire, ou les dates des portées de petits : vous avez besoin d'une autre table. A quoi va t elle ressembler ?

- Elle va devoir contenir les noms des animaux, pour savoir à qui c'est arrivé.
- Il faut une date pour l'événement.
- Il faut un champs de description des événements
- Il faut aussi un champs de catégorie d'événement.

Avec ces indications, la requête de `CREATE TABLE` va ressembler à ceci :

```
mysql62; CREATE TABLE event (nom VARCHAR(20), date DATE,
    −62; type VARCHAR(15), remark VARCHAR(255));
```

Comme pour la table `animaux` table, il est plus facile de charger les premières valeurs à partir d'un fichier, dont les champs sont délimités avec des tabulations :

Chargez les informations comme ceci :

```
mysql62; LOAD DATA LOCAL INFILE "event.txt" INTO TABLE event;
```

Etant donné ce que vous avez appris avec les requêtes sur la table `animaux` table, vous devriez être capable d'exécuter des requêtes sur la table `event`; les principes sont les mêmes. Mais la table `event` pourrait se révéler insuffisante pour répondre à vos questions.

Supposons que vous voulez avoir l'age des animaux lorsqu'ils ont eu leur portée. La table `event` indique quand ils ont eu leur portée, mais pour calculer l'age de la mère, il faut aussi sa date de naissance. Etant donné que cette date est stockée dans la table `animaux`, vous avez besoin des deux tables dans la même requête :

```
mysql62; SELECT animaux.nom, (TO_DAYS(date) − TO_DAYS(naissance))/365 AS age, remarque
    −62; FROM animaux, event
    −62; WHERE animaux.nom = event.nom AND type = "portée";
+--------+------+-------------------------------+
| nom    | age  | remarque                      |
+--------+------+-------------------------------+
| Fluffy | 2.27 | 4 chatons, 3 femelles, 1 male |
| Buffy  | 4.12 | 5 chiots, 2 femelles, 3 male  |
| Buffy  | 5.10 | 3 chiots, 3 femelles          |
+--------+------+-------------------------------+
```

Il faut remarquer plusieurs choses à propos de cette requête :

- La clause FROM est une liste contenant les noms des deux tables, car la requête clause va chercher des informations dans ces deux tables.
- Lorsque vous combinez des informations entre plusieurs tables, il faut spécifier comment les lignes vont correspondre. Ici, la correspondance est simple, et basée sur la colonne nom. La requête utilise une clause WHERE pour rechercher et assortir les valeurs des deux tables, avec la colonne nom.
- Etant donné que les deux tables ont une colonne nom il faut préciser la table d'appartenance de ces colonnes à chaque référence. C'est facilement faisable en ajoutant simplement le nom de la table devant le nom de la colonne.

Les regroupements sont aussi possibles sur une même table. Cela revient à comparer des valeurs d'une table avec d'autres valeurs de la même table. Par exemple, pour marier vos animaux entre eux, vous pouvez faire un regroupement de la table animaux avec elle−même pour rechercher les males et femelles de la même espèce :

```
mysql62; SELECT p1.nom, p1.genre, p2.nom, p2.genre, p1.espece
    −62; FROM animaux AS p1, animaux AS p2
    −62; WHERE p1.espece = p2.espece AND p1.genre = "f" AND p2.genre = "m";
+--------+-------+--------+-------+---------+
| nom    | genre | nom    | genre | espece  |
+--------+-------+--------+-------+---------+
| Fluffy | f     | Claws  | m     | chat    |
| Buffy  | f     | Fang   | m     | chien   |
| Buffy  | f     | Bowser | m     | chien   |
+--------+-------+--------+-------+---------+
```

Dans cette requête, plusieurs alias sont définis pour pouvoir faire référence à chaque instance de la table, et à la bonne colonne.

8.5 Informations sur les bases de données et tables

Que se passe t il si vous oubliez le nom d'une base de données, d'une table ou la structure d'une table donnée ? ***MySQL*** résoud ce problème avec plusieurs commandes qui fournissent des informations sur les bases et les tables.

Vous avez déjà rencontré SHOW DATABASES, qui liste les bases gérés par le serveur. Pour connaître la base de données courante, utilisez DATABASE():

```
mysql62; SELECT DATABASE();
+------------+
| DATABASE() |
+------------+
| menagerie  |
+------------+
```

Si vous n'avez pas de base sélectionnée, le résultat est vide.

Pour connaître les tables de la base de données courante, (par exemple, si vous n'êtes pas sur du nom d'une table), utilisez la commande suivante :

```
mysql62; SHOW TABLES;
+---------------------+
| Tables in menagerie |
+---------------------+
| event               |
| animaux             |
+---------------------+
```

Pour connaître les colonnes d'une table, (par exemple, si vous n'êtes pas sur du nom d'une colonne), utilisez la

commande suivante :

```
mysql62; DESCRIBE animaux;
+---------------+-------------+------+-----+---------+-------+
| Field         | Type        | Null | Key | Default | Extra |
+---------------+-------------+------+-----+---------+-------+
| nom           | varchar(20) | YES  |     | NULL    |       |
| proprietaire  | varchar(20) | YES  |     | NULL    |       |
| espece        | varchar(20) | YES  |     | NULL    |       |
| genre         | char(1)     | YES  |     | NULL    |       |
| naissance     | date        | YES  |     | NULL    |       |
| mort          | date        | YES  |     | NULL    |       |
+---------------+-------------+------+-----+---------+-------+
```

`Field` donne le nom de la colonne, `Type` est le type de données, `Null` indique si la colonne accepte la valeur `NULL` ou pas, `Key` indique que la colonne est manual_tocée, et `Default` indique la valeur par défaut de la colonne.

Si vous avez des manual_toc sur une table, `SHOW manual_toc FROM tbl_nom` fournit la liste des informations les concernant.

8.6 Utiliser `mysql` en mode batch

Dans les sections précédentes, vous avez utilisé `mysql` de manière interactive pour entrer des requête et voir les résultats.Vous pouvez aussi utiliser `mysql` en mode batch. Pour cela, il faut mettre les commandes que vous souhaitez exécuter dans un fichier, puis indiquez à `mysql` qu'il faut l'utiliser comme fichier d'entrée.

```
shell62; mysql 60; batch−file
```

Si vous devez préciser des paramètres de connexions sur la ligne de commande, elle peut ressembler à ceci :

```
shell62; mysql −h host −u user −p 60; batch−file
Enter password: ********
```

Quand vous utilisez `mysql` de cette façon, vous créez un fichier de script, puis exécutez ce script.

Pourquoi utiliser un script ? Voici quelques réponses :

- Si vous exécutez une requête de manière régulière (par exemple, tous les jours, toutes les semaines), utiliser un script vous évitera de tout retaper à la main à chaque fois.
- Vous pouvez générer de nouvelles requêtes à partir d'autre requête déjà existantes, en utilisant le copier/coller.
- Le mode batch peut être pratique lorsque vous développez des requêtes de plusieurs lignes, ou des séquences de requêtes. A chaque erreur, vous n'avez pas tout à retaper, mais juste la commande erronée. Il suffit pour ca d'éditer le script, et de le faire ré exécuter.
- Si une requête génère beaucoup d'informations, vous pouvez les voir avec un outil adapté, plutôt que de les regarder simplement défiler sur l'écran.

```
shell62; mysql 60; batch−file | more
```

- Vous pouvez enregistrer les résultats dans un fichier pour les retraiter ailleurs

```
shell62; mysql 60; batch−file 62; mysql.out
```

- Vous pouvez distribuer le script à d'autres personnes, pour qu'ils puissent exécuter ces commandes aussi.
- Certaines utilisations n'autorisent pas l'utilisation interactif, comme par exemple, la commande cron. Dans ce cas, il faut utiliser le mode batch.

Le format de réponse en mode batch est plus concis qu'en mode intéractif. Par exemple `SELECT DISTINCT espece FROM animaux` ressemble à ceci ,en mode intéractif :

```
+---------+
| espece  |
+---------+
| oiseau  |
| chat    |
| chien   |
| hamster |
| serpent |
+---------+
```

Mais en mode batch, il ressemble à ceci :

```
espece
oiseau
chat
chien
hamster
serpent
```

Pour obtenir un format de réponse " intéractif " à partir du mode batch, utilisez l'option `mysql −t`. Pour avoir aussi les commandes exécutées utilisez l'option `mysql −vvv`.

8.7 Requêtes pour le projet "jumeaux"

Chez Analytikerna and Lentus (NDT : société de l'auteur), nous avons pris en charge l'étude du système et l'architecture des données pour un grand projet de recherche. Ce projet est une collaboration entre l' Institut de médecine environnementale à Karolinska Institutet, Stockholm et la Section Recherche Clinique sur l'age et la psychologie, de l'université de Californie du Sud.

Ce projet avait une grosse partie d'enquête, où tous les jumeaux suédois de plus de 65 ans étaient interviewés par téléphone. Les jumeaux qui remplissaient certains critères étaient admis dans la phase suivante de l'enquête. Dans cette deuxième phase, les jumeaux qui souhaitaient participer, recevaient la visite d'un docteur et d'une infirmière. Une partie des questionnaires étaient des examens physiques et neuropsychologiques, des tests de laboratoire, des scanners du cerveau, des analyses psychologique, et des études généalogiques. De plus, des informations étaient rassemblées sur les risques médicaux et environnementaux.

Pour plus d'information a propos de ce projet, suivez le lien suivant (en anglais) :

```
http://www.imm.ki.se/TWIN/TWINUKW.HTM
```

La deuxième partie du projet est accessible par une interface web, écrite en Perl et ***MySQL***.

Chaque nuit, les informations des réunions étaient insérées dans les base ***MySQL***.

8.7.1 Trouver tous les jumeaux non distribués

8.7.1 Trouver tous les jumeaux non distribués

Les requêtes suivantes sont utilisées pour sélectionner les couples de jumeaux qui accèderont à la deuxième phase :

```
select
```

```
        concat(p1.id, p1.tvab) + 0 as tvid,
        concat(p1.christian_nom, " ", p1.surnom) as Nom,
        p1.postal_code as Code,
        p1.city as City,
        pg.abrev as Area,
        if(td.participation = "Aborted", "A", " ") as A,
        p1.dead as dead1,
        l.event as event1,
        td.suspect as tsuspect1,
        id.suspect as isuspect1,
        td.severe as tsevere1,
        id.severe as isevere1,
        p2.dead as dead2,
        l2.event as event2,
        h2.nurse as nurse2,
        h2.doctor as doctor2,
        td2.suspect as tsuspect2,
        id2.suspect as isuspect2,
        td2.severe as tsevere2,
        id2.severe as isevere2,
        l.finish_date
from
        twin_project as tp
        /* For Twin 1 */
        left join twin_data as td on tp.id = td.id and tp.tvab = td.tvab
        left join informant_data as id on tp.id = id.id and tp.tvab = id.tvab
        left join harmony as h on tp.id = h.id and tp.tvab = h.tvab
        left join lentus as l on tp.id = l.id and tp.tvab = l.tvab
       /* For Twin 2 */
        left join twin_data as td2 on p2.id = td2.id and p2.tvab = td2.tvab
        left join informant_data as id2 on p2.id = id2.id and p2.tvab = id2.tvab
        left join harmony as h2 on p2.id = h2.id and p2.tvab = h2.tvab
        left join lentus as l2 on p2.id = l2.id and p2.tvab = l2.tvab,
        person_data as p1,
        person_data as p2,
        postal_groups as pg
where
        /* p1 gets main twin and p2 gets his/her twin. */
        /* ptvab is a field inverted from tvab */
        p1.id = tp.id and p1.tvab = tp.tvab and
        p2.id = p1.id and p2.ptvab = p1.tvab and
        /* Just the sceening survey */
        tp.survey_no = 5 and
        /* Skip if partner died before 65 but allow emigration (dead=9) */
        (p2.dead = 0 or p2.dead = 9 or
         (p2.dead = 1 and
          (p2.mort_date = 0 or
           (((to_days(p2.mort_date) − to_days(p2.naissanceday)) / 365)
            62;= 65))))
        and
        (
        /* Twin is suspect */
        (td.future_contact = 'Yes' and td.suspect = 2) or
        /* Twin is suspect − Informant is Blessed */
        (td.future_contact = 'Yes' and td.suspect = 1 and id.suspect = 1) or
        /* No twin − Informant is Blessed */
        (ISNULL(td.suspect) and id.suspect = 1 and id.future_contact = 'Yes') or
        /* Twin broken off − Informant is Blessed */
        (td.participation = 'Aborted'
         and id.suspect = 1 and id.future_contact = 'Yes') or
        /* Twin broken off − No inform − Have partner */
        (td.participation = 'Aborted' and ISNULL(id.suspect) and p2.dead = 0))
```

```
        and
        l.event = 'Finished'
        /* Get at area code */
        and substring(p1.postal_code, 1, 2) = pg.code
        /* Not already distributed */
        and (h.nurse is NULL or h.nurse=00 or h.doctor=00)
        /* Has not refused or been aborted */
        and not (h.status = 'Refused' or h.status = 'Aborted'
        or h.status = 'Died' or h.status = 'Other')
order by
        tvid;
```

Quelques explications s'imposent :

- `concat(p1.id, p1.tvab) + 0 as tvid`

On veut trier les valeurs avec la concaténation de `id` et `tvab` dans un ordre numérique. Ajouter `0` au nombre force ***MySQL*** à traiter le résultat comme un nombre

column `id`

Ceci identifiera un couple de jumeaux. C'est une clé commune à toutes les tables.

column `tvab`

Cette colonne identifie un des jumeaux dans un couple. Il prend la valeur de `1` ou `2`.

column `ptvab`

C'est le complémentaire de la colonne précédente. Quand `tvab` vaut `1` celle−ci vaut `2`, et vice versa. Elle sert à éviter des saisie, et permet à ***MySQL*** d'optimiser la requête.

Cette requête démontre, entre autres choses, comment faire des recherches dans une table à partir de la même table, gr ce à un regroupement ((***p1*** et ***p2***). Dans l'exemple ci−dessus, on s'en sert pour vérifier si le deuxième jumeau n'est pas mort avant l'age de 65 ans. Dans ce cas, la ligne n'est pas renvoyée.

Toutes les informations ci−dessus existent dans les tables sur les jumeaux. Il y a toujours une clé sur `id,tvab` (toutes tables) et `id,ptvab` (`person_data`) pour rendre les requêtes plus rapides.

Sur notre serveur de production (une station Sun UltraSPARC 200MHz), cettre requête retourne entre 150 et 200 lignes, et prend moins d'une seconde.

Le nombre courant de ligne dans les tables sont les suivants :

8.7.2 Afficher une table par pair de jumeau

Chaque interview est conclu par un status appelé ***event***. La requête ci−dessous est utilisée pour créer une table avec toutes les paires de jumeaux. Elle indique aussi le nombre de couples qui ont terminé les entretiens, les couples où un seul jumeau a été interrogé, les couples qui ont refusé, etc…

```
select
        t1.event,
        t2.event,
```

```
        count(*)
from
        lentus as t1,
        lentus as t2,
        twin_project as tp
where
        /* We are looking at one pair at a time */
        t1.id = tp.id
        and t1.tvab=tp.tvab
        and t1.id = t2.id
        /* Just the sceening survey */
        and tp.survey_no = 5
        /* This makes each pair only appear once */
        and t1.tvab='1' and t2.tvab='2'
group by
        t1.event, t2.event;
```

9 Serveur MySQL

9.1 Quels sont les langues supportés par MySQL?

`mysqld` peut émettre des messages d'erreur dans les langues suivantes : Hollandais, Anglais (par défaut), Estonien, Français, Allemand, Hongrois, Italien, Norvégien, Polonais, Portugais, Espagnol et Suédois.

Pour démarrer `mysqld` avec une langue donnée, utilisez l'option `--language=lang` ou `-L lang` . Par exemple :

```
shell62; mysqld --language=swedish
```

ou:

```
shell62; mysqld --language=/usr/local/share/swedish
```

Notez que tous les noms de langues sont spécifiés en minuscules.

Le fichie de langue est situé (par défaut, dans le dossier `*mysql_base_dir*/share/*LANGUAGE*/'.

Pour modifier le fichier de message d'erreur, il vous faut éditer le fichier `errmsg.txt' et exécuter la commande suivante pour générer le fichier `errmsg.sys' :

```
shell62; comp_err errmsg.txt errmsg.sys
```

Si vous mettez à jour ***MySQL***, n'oubliez pas de répeter la même manoeuvre avec le nouveau fichier @new `errmsg.txt'.

9.1.1 Le jeu de caractère utilisé pour le tri des données

Par défaut, ***MySQL*** utilise le jeu de caractère ISO−8859−1 (Latin1). C'est le jeu de caractères utilisé aux USA et en Europe Occidentale.

Le jeu de caractères détermine les caractères autorisés dans les noms, et la méthode de tri utilisée dans les clauses `ORDER BY` et `GROUP BY` de la commandes `SELECT`.

Vous pouvez changer le jeu de caractères au moment de la compilation, en utilisant l'option `--with−charset=charset` de `configure`. 4.7.1 Introduction à l'installation rapide.

Pour ajouter un nouveau jeu de caractères à ***MySQL***, utilisez la procédure suivante :

9.1.2 Ajouter un nouveau jeu de caractère

1. Choisir un nom pour le jeu de caractère, dénommé `MYSET`.
2. Créer le fichier `strings/ctype−MYSET.c' dans la distribution source de ***MySQL***.
3. Regarder dans le fichier `ctype−*.c' pour savoir ce qui doit être défini. Notez que les tableaux de votre fichier doivent avoir des noms du type `ctype_MYSET`, `to_lower_MYSET` etc... `to_lower[]` et `to_upper[]` sont de simples tableaux qui contiennent les caractères minuscules et majuscules du jeu de caractère. Par exemple :
   ```
   to_lower['A'] doit contenir 'a'
   to_upper['a'] doit contenir 'A'
   ```

`sort_order[]` est une map qui indique comment les caractères doivent être ordonné lors des comparaisons et des tris. Pour la plus part des jeux de caractères, c'est la même valeur que pour `to_upper[]` (ce qui signifie que le tri sera insensible à la casse). ***MySQL*** effectuera les tris en se basant sur la valeur de `sort_order[character]`. `ctype[]` est un tableau de valeurs de bit, avec un élément par caractère. (Notez que `to_lower[]`, `to_upper[]` et `sort_order[]` sont indexés par valeur de caractère, mais que `ctype[]` est indexé par valeur de caractère +1. C'est une vieille technique qui permet de gérer les EOF). Vous pouvez trouver la définitions de ces champs de bit dans `` `m_ctype.h' ``:

```
#define _U      01      /* Majuscule */
#define _L      02      /* Minuscule */
#define _N      04      /* Chiffre */
#define _S      010     /* Espacement*/
#define _P      020     /* Ponctuation */
#define _C      040     /* Caractère de contrôle */
#define _B      0100    /* Blanc */
#define _X      0200    /* Chiffre hexadécimal */
```

L'entrée de `ctype[]` pour chaque caractère doit être l'union des masques de bits qui décrivent le caractère. Par exemple, `'A'` est une caractère majuscule, (`_U`) mais aussi un chiffre hexadécimal. (`_X`), donc son type devrait contenir `ctype['A'+1]`

```
_U + _X = 01 + 0200 = 0201
```

4. Ajouter un numéro unique pour votre jeu de caractères dans `` `include/m_ctype.h.in' ``.
5. Ajouter le nom du jeu de caractère dans la liste `CHARSETS_AVAILABLE` de `configure.in`.
6. Reconfigurer, recompiler et tester.

9.1.3 Support des caractères multi−byte

Si vous créez un jeu de caractères multi byte, vous pouvez utiliser la macro `_MB`. Dans le fichier `` `include/m_ctype.h.in' ``, ajouter :

```
#define MY_CHARSET_MYSET  X
#if MY_CHARSET_CURRENT == MY_CHARSET_MYSET
#define USE_MB
#define USE_MB_IDENT
#define ismbchar(p, end)  (...)
#define ismbhead(c)       (...)
#define mbcharlen(c)      (...)
#define MBMAXLEN          N
#endif
```

où :

`MY_CHARSET_MYSET`	Une valeur de jeu de caractère unique.
`USE_MB`	Ce jeu de caractère a des caractères multi−byte, géré par `ismbhead()` et `mbcharlen()`
`USE_MB_IDENT`	(optionnel) Si défini, vous pouvez utiliser des noms de tables et de colonnes avec des caractères multi bytes.
`ismbchar(p, e)`	retourne 0 si `p` ne contient pas de caractère multi−byte , ou bien la taille des caractères (en octets). `p` et `e` pointent au début et à la fin de la chaîne. Vérifier de `(char*)p` à `(char*)e−1`.
`ismbhead(c)`	Vrai (Trueà si `c` est le premier caractère d'une chaîne multi−byte.
`mbcharlen(c)`	Taille d'une chaîne multi−byte si `c` est le premier caractère d'une chaîne.
`MBMAXLEN`	Taille en byte du plus grand caractère du jeu de caractère.

9.2 Historique de modification

When started with the `--log-update=file_name` option, `mysqld` writes a log file containing all SQL commands that update data. The file is written in the data directory and has a name of `file_name.#`, where # is a number that is incremented each time you execute `mysqladmin refresh` or `mysqladmin flush-logs`, the `FLUSH LOGS` statement, or restart the server.

If you use the `--log` or `-l` options, the filename is `` `hostname.log' ``, and restarts and refreshes do not cause a new log file to be generated. By default, the `mysql.server` script starts the ***MySQL*** server with the `-l` option. If you need better performance when you start using ***MySQL*** in a production environment, you can remove the `-l` option from `mysql.server`.

Update logging is smart since it logs only statements that really update data. So an `UPDATE` or a `DELETE` with a `WHERE` that finds no rows is not written to the log. It even skips `UPDATE` statements that set a column to the value it already has.

If you want to update a database from update log files, you could do the following (assuming your log files have names of the form `` `file_name.#' ``):

```
shell62; ls -1 -t -r file_name.[0-9]* | xargs cat | mysql
```

`ls` is used to get all the log files in the right order.

This can be useful if you have to revert to backup files after a crash and you want to redo the updates that occurred between the time of the backup and the crash.

You can also use the update logs when you have a mirrored database on another host and you want to replicate the changes that have been made to the master database.

9.3 Taille des tables MySQL

MySQL itself has a 4G limit on table size, and operating systems have their own file size limits. On Linux, the current limit is 2G; on Solaris 2.5.1, the limit is 4G; on Solaris 2.6, the limit is going to be 1000G. Currently, table sizes are limited to either 4G (the ***MySQL*** limit) or the operating system limit, whichever is smaller. To get more than 4G requires some changes to ***MySQL*** that are on the TODO. F Liste de voeux pour les versions futures de MySQL (la TODO).

If your big table is going to be read-only, you could use `pack_isam` to merge and compress many tables to one. `pack_isam` usually compresses a table by at least 50%, so you can have, in effect, much bigger tables. `pack_isam`.

Another solution can be the included MERGE library, which allows you to handle a collection of identical tables as one. (Identical in this case means that all tables are created with identical column information.) Currently MERGE can only be used to scan a collection of tables because it doesn't support indexes. We will add indexes to this in the near future.

10 Améliorer les performances de MySQL

10.1 Optimisation des valeurs du serveur

Vous pouvez accéder aux tailles par défaut des buffers de `mysqld` avec la commande suivante :

```
shell62; mysqld --help
```

Cette commande produit la liste de toutes les options et les variables de `mysqld`. La réponse fournit aussi les valeurs par défaut, et devrait ressembler à ceci :

```
Possible variables for option --set-variable (-O) are:
back_log              current value: 5
connect_timeout       current value: 5
delayed_insert_timeout  current value: 300
delayed_insert_limit  current value: 100
delayed_queue_size    current value: 1000
flush_time            current value: 0
join_buffer_size      current value: 131072
key_buffer_size       current value: 1048540
long_query_time       current value: 10
max_allowed_packet    current value: 1048576
max_connections       current value: 100
max_connect_errors    current value: 10
max_delayed_threads   current value: 20
max_heap_table_size   current value: 16777216
max_join_size         current value: 4294967295
max_sort_length       current value: 1024
max_tmp_tables        current value: 32
net_buffer_length     current value: 16384
record_buffer         current value: 131072
sort_buffer           current value: 2097116
table_cache           current value: 64
tmp_table_size        current value: 1048576
thread_stack          current value: 131072
wait_timeout          current value: 28800
```

Si vous avez un serveur `mysqld` en fonctionnement, vous pouvez voir les valeurs réellement utilisées en exécutant la commande suivante :

```
shell62; mysqladmin variables
```

Toutes les options dont détaillées ci-dessous. Les tailles des buffers et des piles, les longueurs sont toutes données en octets. Vous pouvez ajouter le suffixe ``K'' ou ``M'' pour indiquer kilo-octets or mégaoctets. Par exemple, `16M` représente 16 mégaoctets. La casse du suffixe n'a pas d'importance, et; `16M` et `16m` sont équivalents.

- back_log Le nombre maximum de tentative de connexions simultanées au serveur ***MySQL*** . Cette option entre en jeu lors que le thread principal ***MySQL*** reçoit un très ***GRAND*** nombre de connexion dans un délai très bref. Le thread principal requiert un peu de temps (même si c'est très peu) pour vérifier la connexion, et lancer un nouveau thread. L'option `back_log` indique le nombre de connexion qui sont mise en attente, avant que ***MySQL*** momentanément cesse de réponse à de nouvelles requêtes. Augmentez la valeur pour accepter un grand nombre de connexions dans un délais très court. En d'autres termes, cette valeur correspond à la taille de la file d'attente de connexions TCP/IP. Votre système d'exploitation a sa propre limite. Sous Unix, le manuel donne plus de détails avec l'entrée `listen(2).` Vérifier la documentation de votre OS pour connaître la valeur de cette variable. Affecter à `back_log` une valeur plus grande que le maximum accepté par le système sera inefficace.
- connect_timeout Le nombre de secondes d'attente d'un paquet de connexion avant que `mysqld` ne réponde une erreur `Bad handshake`.
- delayed_insert_timeout Le temps d'attente d'un thread `INSERT DELAYED` : si les commandes `INSERT` sont trop longues, le thread `INSERT`

DELAYED se finira.
- delayed_insert_limit Après avoir inséré delayed_insert_limit lignes, le handler INSERT DELAYED vérifiera qu'il n'y a aucune commande SELECT en attente. Si c'est le cas, le handler INSERT DELAYED sera exécuté avant de continuer.
- delayed_queue_size La taille de la queue d'attente qui gère les INSERT DELAYED. Si la queue est pleine, tout client qui émet une commande INSERT DELAYED devra attendre qu'une place se libère dans la queue.
- flush_time Si cette option a une valeur, alors, toutes les flush_time secondes, toutes les tables seront fermées.(pour libérer des ressources, et synchroniser les tables sur le disque).
- join_buffer La taille des buffers utilisés lors des regroupements sans index (les plus gourmands en place). Le buffer est alloué à chaque regroupement entre deux tables. Augmentez cette valeur pour obtenir un regroupement plus rapide lorsque l'ajout d'index n'est pas possible (ce qui est la méthode la plus efficace pour accélérer une requête).
- key_buffer Les blocs d'index sont bufferisés et partagé par tous les threads. key_buffer est la taille des buffers de bloc d'index. Cela permettra d'accélerer le traitement des commandes DELETE ou INSERT sur une table avec de nombreux index. Pour accélerer encore plus la commande, reportez vous à la section LOCK TABLES
- long_query_time Si une requête requiert plus que time secondes, le compteur de slow_queries sera incrémentés.
- max_allowed_packet La taille maximale d'un paquet. Ce buffer est initialisé à la longueur net_buffer_length octets, mais il peut croître jusqu'à une taille de max_allowed_packet octets, au besoin. Cette valeur par défaut est trop petite pour gérer de gros paquets (et probablement ceux−ci sont érronnés). Il vaut mieux augmenter la taille cette colonne pour pouvoir gérer des objets de BLOB : il faudrait que la valeur de cette option soit aussi grande de le plus grand BLOB que vous voulez utiliser.
- max_connections Le nombre maximal de connexion simutanée de client. En augmentant cette valeur, vous augmenter le nombre de pointeur de fichier que requiert mysqld . Voir ci−après pour plus de détails sur les pointeurs de fichiers
- max_connect_errors S'il y a plus que max_connect_errors connexions interrompues de la part d'un même hote, toutes les tentatives ultérieures de connexions seront bloquées. Vous pouvez aussi débloquer les hôtes avec la commande FLUSH HOSTS.
- max_delayed_threads Force le serveur à limiter le nombre de handle INSERT DELAYED à max_delayed_threads. Si vous tentez d'insérez des informations dans une nouvelle table après que tous les threads INSERT DELAYED soient utilisés, l'attribut DELAYED sera ignoré.
- max_join_size Les regroupements qui vont probablement rassembler plus de max_join_size lignes retourne une erreur. Affectez une limite si vos utilisateurs prennent la mauvaise habitude d'exécuter des regroupements de plusieurs millions de lignes.
- max_sort_length Le nombre d'octets à utiliser pour classer les objets de type BLOB ou TEXT (Seuls lest max_sort_length premiers octets de chaque valeurs seront utilisés, le reste sera ignoré).
- max_tmp_tables (Cette option n'est pas encore effective). Nombre maximum de tables temporaires qu'un client peut avoir en même temps.
- net_buffer_length Le buffer de communication est redimensionné entre deux requête. Cette option ne devrait généralement pas être modifiée, mais si vous avez très peu de mémoire, vous pouvez lui affecter la taille attendue de la requête, c'est à dire la longueur des commandes envoyées par les clients). Si une commande est trop grande, le buffer sera automatiquement étendu jusqu'à max_allowed_packet octets.
- record_buffer Chaque thread qui effectue une analyse séquentielle, alloue un buffer de cette taille pour chaque table scannée. Pour augmenter le nombre de recherche séquentielles simultanées, augmentez cette valeur.
- sort_buffer Chaque thread qui a besoin de classer des données alloue un buffer de sort_buffer octets. Augmentez cette valeur pour accélerer les clauses ORDER BY ou GROUP BY. 18.5 Où MySQL enregistre les fichiers temporaires
- table_cache Le nombre de tables ouvertes par tous les threads. Augmenter cette valeur revient à augmenter le nombre de pointeur de fichier dont mysqld a besoin. ***MySQL*** a besoin de deux pointeurs de fichiers pour chaque table. Voir ci−dessous les remarques à propos de la limite du nombre de pointeur de fichier. Pour des détails concernant le cache des tables, 10.8 Comment MySQL ouvre et ferme les tables.
- tmp_table_size Si une table temporaire excède cette taille, ***MySQL*** génère une erreur sous la forme : The table nom_table is full. Augmentez la valeur de tmp_table_size pour réaliser des requêtes GROUP BY plus compliquées.
- thread_stack La taille de la pile pour chaque thread. La plus part du temps, les erreurs repérées par crash−me dépendent de cette valeur. La valeur par défaut est généralement suffisante. Benchmarks.
- wait_timeout Le nombre de seconde d'inactivité d'une connexion avant qu'elle soit fermée.

table_cache, max_connections et max_tmp_tables limite le nombre maximal de fichier que le serveur peut avoir ouvert en même temps. Si vous augmentez ces valeurs, vous risquez d'être limité par votre système d'exploitation. Cependant, il est possible d'augmenter le nombre de pointeur de fichier par processus, sur certains systèmes. Consultez la documentation de votre système, car ces manipulations varient beaucoup de l'un à l'autre. table_cache est lié à max_connections. Par exemple, pour 200 connexions, vous devez avoir un cache de table d'au moins 200 * n, avec n nombre maximum de tables dans un regroupement. ***MySQL*** utilise des algorithmes très efficaces, ce qui permet de le faire tourner avec peut de mémoire. Cependant, pour accélérer les traitement, allouez plus de mémoire à MySQL. En ayant beaucoup de mémoire, et beaucoup de table, si vous souhaitez de bonne performances, avec un nombre modéré de clients, utilisez une configuration telle que :

```
shell62; safe_mysqld −O key_buffer=16M −O table_cache=128 \
           −O sort_buffer=4M −O record_buffer=1M 38;
```

Si vous avez peu de mémoire et beaucoup de connexion, utilisez une configuration telle que :

```
shell62; safe_mysqld −O key_buffer=512k −O sort_buffer=100k \
           −O record_buffer=100k 38;
```

voire même :

```
shell62; safe_mysqld −O key_buffer=512k −O sort_buffer=16k \
           −O table_cache=32 −O record_buffer=8k −O net_buffer=1K 38;
```

Si le nombre de connexion est vraiment très grand, le système va commencer à swapper, à moins que `mysqld` n'ait été configuré pour utiliser très peu de mémoire pour chaque connexion. Bien entendu, `mysqld` fonctionne bien mieux si vous avez assez de mémoire pour toutes les connexions.

NB : si vous changez une option de `mysqld`, cette dernière ne sera prise en compte qu'à la prochaine instance du serveur.

Pour voir si les changement de paramètres ont été pris en compte, utilisez la commande :

```
shell62; mysqld −O key_buffer=32m −−help
```

Assurez vous que l'option `−−help` est mise en dernier, sinon les options placées après seront ignorées.

10.2 Comment MySQL gère la mémoire

La liste suivante indique la manière dont `mysqld` utilise la mémoire. A chaque fois que c'est possible, le nom des variables systèmes adéquates sont cités.

- Le buffer de clé (variable `key_buffer`) est partagée par tous les threads; Les autres buffers sont alloués selon les nécessités.
- Chaque connexion utilise un thread particulier, une pile (64K, variable `thread_stack`) ; un buffer de connexion (variable `net_buffer_length`), et un buffer de résultat (variable `net_buffer_length`). Les buffers de connexion et de résultat sont dynamiquement aggrandi jusqu'à la taille maximale de `max_allowed_packet` selon les besoins. Quand une requête est en cours d'exécution, une copie de la chaîne de requête est aussi allouée.
- Tous les threads se partage la mémoire de base.
- Hormis les tables compressées, rien n'est mappé en mémoire actuellement. Cela provient du fiat que les adresse de 32bits, soit 4 Go ne sont pas assez grand pour la plus part des très grande tables. Dès qu'un système d'exploitation avec des adresses de 64−bit sera disponible, nous pourrons ajouter le mappage mémoire.
- Chaque requête qui fait une recherche séquentielle créer un buffer de lecture (variable `record_buffer`).
- Tous les regroupements sont fait en une seule passe, et la plus part des regroupements peut être fait sans avoir recours à une table temporaire. La plus part des tables temporaires sont gardée en mémoire vive (HEAP). Les tables temporaires avec des enregistrements de grandes tailles (calculées avec la somme des longueurs de colonnes) ou qui contiennent des objets de type `BLOB` sont stockées sur le disque. Un des problèmes récurent est que si une table stockées en HEAP dépasse la taille de `tmp_table_size`, vous obtener une erreur de type `The table nom_table is full`. Dans les prochaines versions, les tables temporaires sauvées en mémoire seront automatiquement transférées sur le disque. Pour contourner le problème, vous pouvez augmenter le nombre de tables temporaires avec l'option `tmp_table_size` ou avec l'option SQL `SQL_BIG_TABLES` dans le programme client. `SET OPTION`. Avec ***MySQL*** 3.20, la taille maximum des tables temporaires était `record_buffer*16`, donc, si vous utilisez cette version, vous devez augmenter la valeur de `record_buffer`. Vous pouvez aussi démarrer `mysqld` avec l'option `−−big−tables` pour forcer le stockarge des tables temporaires sur le disque. Cependant, cela va affecter les performances des requêtes compliquées.
- La plus part des requêtes qui font un tri, alloue un buffer de tri, et un ou deux fichiers temporaires. 18.5 Où MySQL enregistre les fichiers temporaires.
- Toutes les analyses et les calculs sont fait en mémoire vive. Aucune mémoire supplémentaire n'est nécessaire pour les petits objets, et les problèmes d'allocations et dedésallocation de mémoire lente sont évités. On utilise alors ces méthodes uniquement pour les chaînes les plus longues.
- Chaque fichier d'index est ouvert une fois, et chaque fichier de données est ouvert par chaque thread concurent. Pour chacun des threads, une table de structure (colonnes et structure de chaque colonne) et un buffer de 3 * n est alloué, avec `n` la taille maximum d'une ligne, hormis les `BLOB`). Un objet `BLOB` ustilise 5 à 8 octets plus la taille des données de l'objet `BLOB`.
- Pour chaque table avec des colonnes `BLOB`, un buffer est automatiquement aggrandi pour pouvoir lire les grands objets `BLOB`. Si vous scannez une table, un buffer aussi grand que le plus grand des objets `BLOB` sera alloué.
- Les handlers de table pour les tables couramment ouvertes sont sauvés dans un cache, et géré comme une pile de type FIFO. Généralement, ce cache a 64 entrées. Si une table a été utilisée par deux threads concurents en même temps, le cache contient deux fois la table. 10.8 Comment MySQL ouvre et ferme les tables.
- Une commande `mysqladmin flush−tables` ferme toutes les tables qui ne sont pas en cours d'utilisation, et s'assure que toutes les tables ouvertes seront aussitôt refermées dès que le thread qui les utilisent auront terminé. Cela permettra de libérer l'essentiel de la mémoire.

ps et d'autres programmes rapportent parfois que `mysqld` utilise beaucoup de mémoire. Cela peut être du à de nombreuses piles, dans différents threads, à différentes adresses. Par exemple, la version Solaris de `ps` compte la mémoire inutilisée entre les piles comme de la mémoire utilisée. Vous pouvez vérifier ceci en vérifiant le swap disponible, avec : `swap −s`. Nous avons testé `mysqld` avec des détecteurs de fuite de mémoire du commerce, et il n'y a plus de fuites de mémoire.

10.3 Comment la compilation et le link affecte la vitesse de MySQL

La plus part des tests suivants ont été menés sous Linux, et avec la suite de test de ***MySQL***, mais les résultats doivent être transposables aux autres systèmes d'exploitation.

L'exécutable le plus rapide est obtenu avec l'option de link `−static`. Utiliser des sockets Unix plutôt que TCP/IP pour se connecter à la base de données est aussi plus efficace.

Avec Linux, l'exécutable le plus rapide fut obtenu en compilant avec `pgcc` et `−O6`. Pour compiler ``sql_yacc.cc'' avec ces options, vous aurez besoin de 180Mo de méoire, car `gcc/pgcc` requiert beaucoup de mémoire pour rendre toutes les fonctions inline. Vous devriez aussi utiliser l'option `CXX=gcc` lors de la configuration de ***MySQL*** pour éviter l'inclusion de la bibliothèque `libstdc++` .

- Si vous utilisez `pgcc` et compilez tout avec `−O6`, le serveur ***MySQL*** sera 11% plus rapide qu'avec `gcc`.
- En faisant un link dynamique (sans `−static`), le résultat est 13% plus lent.
- Si vous vous connectez avec TCP/IP plutôt qu'avec les sockets Unix, l'ensemble est 7.5% plus lent.
- Sur une station Sun SPARC Station 10, `gcc` 2.7.3 est 13% plus rapide que Sun Pro C++ 4.2.
- Avec Solaris 2.5.1, MIT−pthreads est 8−12% plus lent que les thread natifs Solaris.

La distribution ***MySQL***−Linux fournie pas TcX est compilée avec `pgcc` et linkée statiquement.

10.4 Utilisation des index

Tous les index (`PRIMARY`, `UNIQUE` et `INDEX()`) sont stockés dans des B−trees. Les chaînes sont automatiquement compressées : `CREATE INDEX`.

Les index sont utilisés pour :

- Rechercher rapidement une ligne qui vérifie une clause `WHERE`.
- Retrouver des lignes d'une autre table lors d'un regroupement.
- Trouver la valeur `MAX()` ou `MIN()` d'une clé donnée
- Trier ou regrouper les éléments d'une table, si le tri ou le regroupement est fait avec le préfixe d'une clé accessible (e.g. `ORDER BY key_part_1,key_part_2`). La clé sont lus dans l'autre sens si on utilise l'attribue `DESC`.
- Retourner des valeurs sans consutler le fichier de données, dans certains cas. Si toutes les colonnes d'une table sont des numériques, et forment la partie gauche du préfixe d'une clé, ces valeurs peuvent être retournée directement depuis un index, pour plus de rapidité.

Supposons que vous voulez exécuter la commande `SELECT` suivante :

```
mysql62; SELECT * FROM nom_table WHERE col1=val1 AND col2=val2;
```

Si un index multi−colonne existe avec `col1` et `col2`, les lignes adéquates peuvent être retrouvée directement. Si des index existe sur les colonnes `col1` et `col2`, l'optimiseur décide quel index va retourner le moins de lignes, et utilise cet index pour retourner les lignes.

Si la table a des index multi−colonnes, tous les prefixes à gauches de l'index peut être utilisé par l'optimiseur pour retrouver les lignes. Par exemple, si vous avez un index de trois colonnes `(col1,col2,col3)`, vous

avez la possibilité de rechercher rapidement sur les colonnes (col1), (col1,col2) et (col1,col2,col3).

MySQL ne peut pas utiliser un index partiel si les colonnes ne forment pas la partie gauche d'un préfixe d'index. Supposons que vous ayez la commande suivante :

```
mysql62; SELECT * FROM nom_table WHERE col1=val1;
mysql62; SELECT * FROM nom_table WHERE col2=val2;
mysql62; SELECT * FROM nom_table WHERE col2=val2 AND col3=val3;
```

Si un index existe sur les colonnes (col1,col2,col3), alors seule la première requête utilisera un index. Les deux autres requête font appel à des colonnes indexées, mais (col2) et (col2,col3) ne sont pas les préfixes à gauche de (col1,col2,col3).

MySQL utilise aussi des index pour des comparaisons de type LIKE si l'argument de LIKE est une chaîne constante qui ne commence pas par un caractère spécial. Par exemple, la commande suivante SELECT utilise des index :

```
mysql62; select * from nom_table where key_col LIKE "Patrick%";
mysql62; select * from nom_table where key_col LIKE "Pat%_ck%";
```

Dans la première commande, seule les lignes "Patrick" <= key_col < "Patricl" sont prises en considération. Dans la deuxième commande, seule les lignes avec "Pat" <= key_col < "Pau" sont prises en considération.

Les commandes suivantes ne vont pas utiliser d'index :

```
mysql62; select * from nom_table where key_col LIKE "%Patrick%";
mysql62; select * from nom_table where key_col LIKE other_col;
```

Dans la première, la clause LIKE commence par un caractère spécial. Dans la deuxième, la clause LIKE n'est pas constante.

Effectuer des recherches en utilisant column_name IS NULL utilisera des index si column_name est un index.

MySQL utilise prioritairement des index qui vont trouver le minimum de ligne. Un index est utilisé lors des comparaisons qui impliquent une colonne et un opérateur =, >, >=, <, <=, BETWEEN ou LIKE sans caractère spécial comme 'quelquechose%'.

Un index qui ne comprend pas toutes les colonnes utilisées dans une clause WHERE avec un opérateur AND n'est pas utilisé pour optimiser la requête.

Les requêtes suivantes utilisent des index :

```
... WHERE index_part1=1 AND index_part2=2
... WHERE index=1 OR A=10 AND index=2      /* index = 1 OR index = 2 */
... WHERE index_part1='hello' AND index_part_3=5
          /* optimized like "index_part1='hello'" */
```

Les requêtes suivantes n'utilisent pas d'index :

```
... WHERE index_part2=1 AND index_part3=2  /* index_part_1 is not used */
```

```
... WHERE index=1 OR A=10                 /* No index */
... WHERE index_part1=1 OR index_part2=10  /* No index spans all rows */
```

10.5 Comment MySQL optimise les clauses WHERE

(Cette section est incomplète, car ***MySQL*** effectue de nombreuses optimisations.)

En général, lorsque vous voulez accélérer le traitement d'une commande SELECT ... WHERE, le premier reflexe a avoir est de regarder si il n'est pas possible d'ajouter un index. Toutes les références entre tables devraient utiliser des index. Vous pouvez utiliser la commande EXPLAIN déterminer quels index sont utilisés au cours d'une commande SELECT. EXPLAIN.

Une partie des optimisations de ***MySQL*** sont présentées ci−dessous :

- Elimination des parenthèses inutiles

```
   ((a AND b) AND c OR (((a AND b) AND (c AND d))))
−62; (a AND b AND c) OR (a AND b AND c AND d)
```

- Remplacement des variables par des constantes

```
   (a60;b AND b=c) AND a=5
−62; b62;5 AND b=c AND a=5
```

- Elimination des conditions constantes (à cause du remplacement des constantes)

```
   (B62;=5 AND B=5) OR (B=6 AND 5=5) OR (B=7 AND 5=6)
−62; B=5 OR B=6
```

- Les expressions constantes utilisées par les index ne sont évaluées qu'une seule fois.
- COUNT(*) exécuté sur une seule table sans clause WHERE est directement lu dans les informations de la table. C'est aussi vrai pour les expression NOT NULL utilisé sur une seule table.
- Détection précoce des expressions constantes invalides. ***MySQL*** détecte que certaines commandes SELECT sont impossibles et retourne aucune ligne.
- HAVING est fusionné avec WHERE si vous n'utilisez pas GROUP BY ou des fonctions de groupage, telles que COUNT(), MIN()...
- Pour chaque sous−regroupement, une requête WHERE plus simple est constituée, pour accélérer l'exécution et abandonner les requêtes les plus tôt possible.
- Toutes les tables constantes sont lues en premier, avant toute autre tables dans la requête. Une table constante est:
  - Une table vide ou avec une seule ligne
  - Une table qui est utilisé avec une clause WHERE avec un index UNIQUE ou une PRIMARY KEY, et si toutes les membres de l'index sont utilisé avec des expressions constantes.

Toutes les tables suivantes sont utilisées comme des tables constantes :

```
mysql62; SELECT * FROM t WHERE primary_key=1;
mysql62; SELECT * FROM t1,t2
           WHERE t1.primary_key=1 AND t2.primary_key=t1.id;
```

- La meilleure combinaison pour un regroupement est réalisée en essayant toutes les possibilités (si toutes les colonnes de la clause ORDER BY et GROUP BY proviennent de la même table, alors cette table est traitée en premier, lors du regroupement) .
- Si il y a une clause ORDER BY et une clause GROUP BY différente, ou si ORDER BY ou GROUP BY contiennent des colonnes d'autres tables que la première table dans la commande de regroupement, une table temporaire est créée.
- Si vous utilisez SQL_SMALL_RESULT, ***MySQL*** utilisera une table temporaire en mémoire.
- Puisque DISTINCT est convertie en GROUP BY sur toutes les colonnes, DISTINCT combiné avec ORDER BY nécessitera la plus part du temps une table temporaire.
- Les index de table sont d'autant plus rapide et efficace qu'ils ne prennent en compte que 30% de la longueur de la ligne. Si aucun index n'est trouvé, la table est rapidement analysée.
- Dans certains cas, ***MySQL*** peut lire des lignes sans même consulter le fichier de données. Si toutes les colonnes utilisées pour l'index sont numériques, alors seul l'arbre d'index sera utilisé pour résoudre la requête.
- Avant d'être retourné, toute ligne qui ne satisfait par le critère de HAVING sont ignorés.

Quelques exemples de requête très rapides :

```
mysql62; SELECT COUNT(*) FROM nom_table;
mysql62; SELECT MIN(key_part1),MAX(key_part1) FROM nom_table;
mysql62; SELECT MAX(key_part2) FROM nom_table
           WHERE key_part_1=constant;
mysql62; SELECT ... FROM nom_table
           ORDER BY key_part1,key_part2,... LIMIT 10;
mysql62; SELECT ... FROM nom_table
           ORDER BY key_part1 DESC,key_part2 DESC,... LIMIT 10;
```

Les requête suivantes sont résolues en utilisant uniquement l'arbre d'index (on supposera que toutes les colonnes indexées sont numériques)

```
mysql62; SELECT key_part1,key_part2 FROM nom_table WHERE key_part1=val;
mysql62; SELECT COUNT(*) FROM nom_table
           WHERE key_part1=val1 AND key_part2=val2;
mysql62; SELECT key_part2 FROM nom_table GROUP BY key_part1;
```

Les requêtes suivantes sont indexées pour retourner les lignes classées, sans nécessiter de clause de classement :

```
mysql62; SELECT ... FROM nom_table ORDER BY key_part1,key_part2,...
mysql62; SELECT ... FROM nom_table ORDER BY key_part1 DESC,key_part2 DESC,...
```

10.6 Comment MySQL optimise les `LEFT JOIN`

Avec MySQL, les `commanndes LEFT JOIN B` sont implémentées comme suit :

- La table `B` est marquée comme étant dépendante de la table `A`.
- La table `A` est marquée comme étant dépendante de toutes les tables (sauf `B`) qui sont utilisées dans la condition `LEFT JOIN`.
- Toutes les conditions `LEFT JOIN` sont transformées en clause `WHERE`.
- Toutes les optimisations standard de regroupement sont effectuées, à l'exception près qu'une table n'est lue qu'après toutes les tables dont elle dépend. Si il y a une référence circulaire ***MySQL*** retournera une erreur.
- Toutes les optimisations standard de `WHERE` sont effectuées.
- Si une ligne de la table `A` satisfait la clause `WHERE`, mais qu'il n'y a aucuen ligne dans la table `B` pour satisfaire la condition `LEFT JOIN`, alors une nouvelle ligne est générée dans `B` avec toutes ses valeurs mises à `NULL`.
- Si vous utilisez un `LEFT JOIN` pour rechercher des lignes qui n'existent pas dans uen table, et que vous avez le test suivant : `column_name IS NULL` dans la clause `WHERE`, où column_name est une colonne déclarée `NOT NULL`, alors `MySQL` arrêtera de cherche d'autres lignes (pour une combinaison particulière de clés) dès qu'il aura trouvé une ligne qui satisfait la condition `LEFT JOIN`.

10.7 Comment MySQL optimise les `LIMIT`

Dans certains cas, ***MySQL*** va traiter la requête de manière très différente suivant que vous utilisez `LIMIT #` et non pas `HAVING`:

- Si vous ne faites que selectionner que quelques lignes avec `LIMIT`, ***MySQL*** utilisera des index, alors qu'il préfère généralement une recherche exhaustive dans la table.
- Si vous utiliser `LIMIT #` avec `ORDER BY`, ***MySQL*** arrêtera le tri dès qu'il a trouver un nombre suffisant de colonnes, et non pas après avoir scanné la table complète.
- Lors de la combinaison de `LIMIT #` et `DISTINCT`, ***MySQL*** s'arrête dès qu'il a trouvé # lignes uniques.
- Dès que `MySQL` a envoyé les premières # au client, il termine la requête.

10.8 Comment MySQL ouvre et ferme les tables

Le cache des tables ouvertes peut grossir jusqu'au maximum de `table_cache` (par défaut 64; mais cela peut être changé avec l'option `-O table_cache=#` de `mysqld`). Une table n'est jamais fermé, sauf si le

cache est plein, et qu'un thread essaie d'ouvrir une autre table, ou si vous utilisez `mysqladmin refresh` ou `mysqladmin flush−tables`.

Lorsque le cache de table ouverte se remplit, le serveur utilise les règles suivantes pour localiser la prochaine entrée à utiliser :

- Les tables qui ne sont pas utilisées sont fermées : MySQL prendra alors en priorité la table qui a été utilisée pour la dernière fois il y a le plus longtemps.
- Si le cache se remplit et qu'aucune table ne peut être fermée, mais qu'une nouvelle table doit être ouverte, le cache est temporairement agrandi en fonction des besoins.
- Si le cache est déjà dans un état étendu, et qu'une table passe de l'état d'utilisation à l'état de non utilisation (tout en restant ouverte), elle est automatiquement fermée, et retirée du cache.

Une table est ouverte à chaque accès concurrent. Cela signifie que si vous avez deux threads qui accèdent en même temps à une table, ou accèdent deux fois à une table dans la même requête (avec AS), la table sera ouverte deux fois. La première ouverture d'une table requière deux pointeurs de fichiers, et chaque accès supplémentaire n'en demande qu'un de plus. Le deuxième pointeur de fichier ouvert lors de la première ouverture est pour le fichier d'index, et de pointeur est partagé entre tous les autres threads.

10.8.1 Problèmes soulevés par un trop grand nombre de table dans des bases

Si vous avez trop de fichier dans un dossier, ouvrir, fermer et évaluer des opérations se fera lentement. Si vous exécutez une commande `SELECT` sur de nombreuses tables, il ne restera plus beaucoup de marge de man§uvre lorsque le cache sera plein, et cela ralentira nettement les exécutions, puisque pour chaque table à ouvrir, il faut qu'une autre se libère. Vous pouvez réduire ces latences en augmentant la taille du cache.

10.9 Pourquoi tant de tables sont elles ouvertes?

Lorsque vous exécutez `mysqladmin status`, vous verrez quelque chose comme ça :

```
Uptime: 426 Running threads: 1 Questions: 11082 Reloads: 1 Open tables: 12
```

Ce qui peut être étrange, surtout si vous n'avez que 6 tables.

MySQL est multi−threadé, ce qui fait que plusieurs requêtes peuvent accéder à la même table en même temps. Pour simplifier le problème, prenons deux threads dans deux états différents, qui accèdent au même fichier. La table est alors ouverte deux fois. Cela requiert de la mémoire et un pointeur de fichier de plus pour le fichier de données. Le pointeur de fichier d'index est partagé par tous les threads.

10.10 Utilisation de liens symboliques pour les bases de données et tables

Vous pouvez déplacer des tables et des bases de données depuis le dossier des bases de données vers n'importe quelle autre place, et les remplacer par des liens symboliques. Cela permet d'implanter les bases dans des systèmes avec plus de place.

Lorsque ***MySQL*** remarque qu'une table est un lien symbolique, il va le résoudre et utiliser à la place la table. Cela fonctionne sur tous les systèmes qui acceptent l'appel système `realpath()` (par exemple : Linux Solaris supportent `realpath()`)! Sur tous les système qui ne supportent pas `realpath()`, vous ne devez pas accéder à la table via le chemin et via le lien en même temps. Si vous le faites, la table sera laissée dans

un état incohérent.

Par défaut, MySQL n'accepte pas les liens symboliques de bases de données. Les choses tournent mal lorsque vous commencez à faire des liens symboliques entre les bases de données. Supposons que vous ayez une base de `db1` dans le dossier de données de ***MySQL***, et que vous créez un lien symbolique `db2` qui pointe sur `db1`:

```
shell62; cd /path/to/datadir
shell62; ln −s db1 db2
```

Maintenant, toute table `tbl_a` de `db1` est aussi une table `tbl_a` de `db2`. Si un thread met à jour `db1.tbl_a` et un autre thread met à jours `db2.tbl_a`, il va y avoir des problèmes de cohérence.

Si vous avez vraiment besoin de cette fonctionnalité, vous devrez changer le code suivant dans ``mysys/mf_format.c'':

```
if (!lstat(to,38;stat_buff))  /* Check if it's a symbolic link */
    if (S_ISLNK(stat_buff.st_mode) 38;realpath(to,buff))
```

Remplacez le par :

```
if (realpath(to,buff))
```

10.11 Comment MySQL verrouille les tables

La gestion des verrous sous ***MySQL*** est inblocable. Cela est réalisé en demandant toujours tous les droits, en même temps, au début de la requête, et en verrouillant toujours les tables dans le même ordre.

La méthode de verrouillage pour les verrous en écriture (`WRITE`) est la suivante :

- Si il n'y a pas de verrous sur la table, ***MySQL*** la verrouille.
- Sinon, il ajoute une demande de verrous dans la queue d'attente.

La méthode de verrouillage pour les verrous en lecture (`READ`) est la suivante :

- Si il n'y a pas de verrous sur la table, MySQL la verrouille.
- Sinon, il ajoute une demande de verrous dans la queue d'attente.

Quand un verrou est libéré, le verrou est rendu disponible pour les threads de la queue d'attente d'écriture, puis dans la queue d'attente de lecture.

Cela signifie que si vous avez de nombreuses modifications sur une table, les commandes `SELECT` devront attente qu'il n'ya ait plus de modifications en attente pour être satisfaites.

Pour éviter ce genre de problème, vous pouvez rassembler toutes vos insertions dans une table temporaire, et faire une mise à jour importante de toute la table temporaire dans la fixe, une fois de temps en temps. Cela se réalise avec le code suivant :

```
mysql62; LOCK TABLES real_table WRITE, insert_table WRITE;
mysql62; insert into real_table select * from insert_table;
mysql62; delete from insert_table;
mysql62; UNLOCK TABLES;
```

Vous pouvez utiliser l'option `LOW_PRIORITY` ou `HIGH_PRIORITY` avec `INSERT` si vous voulez exploiter les priorités. `INSERT.`

Vous pouvez aussi changer la procédure de verrouillage dans le fichier ``mysys/thr_lock.c'' pour n'utiliser qu'une seule queue d'attente. Danx ce cas, les verrous d'écriture et de lecture auront les mêmes priorités, ce qui peut être utile dans certains cas.

10.12 Comment constituer une table pour qu'elle soit petite et rapide?

Vous pouvez accélerer le traitement et minimiser l'espace de stockage d'une table en utilisant les techniques suivantes :

- Déclarez mes colonne comme `NOT NULL` si possible. Cela accélère l'ensemble, et vous économisez un bit par colonne
- Profitez du fait que toutes les colonnes ont une valeur par défaut. Insérez les valeurs explicitement uniquement lorsque la valeur insérée diffère de la valeur par défaut. Vous n'avez ainsi pas besoin d'affecter de valeur à la première colonne de type `TIMESTAMP` ou de type `AUTO_INCREMENT` lors d'une commande `INSERT`. `mysql_insert_id().`
- Utilisez autant que possible la plus petite taille d'entier. Les tables en seront réduites d'autant. Par exemple, `MEDIUMINT` est souvent plus efficace que `INT`.
- Si vous n'avez pas de colonne de longueur variable (`VARCHAR`, `TEXT` ou `BLOB`), la ligne est de longueur fixe. Cela nettement plus rapide, mais cela peut gaspiller un peu de place . 10.17 Quel sont les différents formats de lignes? Quand utiliser `VARCHAR/CHAR`?.
- Pour aider ***MySQL*** à optimisez les requêtes, utilisez `isamchk --analyze` sur une table, après l'avoir chargée avec les données adéquates. Cela met à jours une caractéristiques de la table, qui indique le nombre moyen de ligne qui ont la même valeur. (Pour les index uniques, cette valeur vaudra 1, bien entendu).
- Pour classer un index et des données en fonction d'un index, utilisez `isamchk --sort-index --sort-records=1` (pour trier avec l'index 1). Si vous avez un index unique dont vous voulez lire toutes lignes dans l'ordre de l'index, c'est une bonne idée pour accélèrer la requête. Remarquez bien que ce classement n'est pas optimal, et qu'il peut s'avérer long pour une grande table.
- Pour les commandes `INSERT`, utilisez des insertions multiples. C'est beaucoup plus rapide que de faire des insertions séparées.
- Lorsque vous chargez des données, utilisez `LOAD DATA INFILE`. Cette commande est généralement 20 fois plus rapide que des commandes `INSERT` repétées. `LOAD DATA`. Vous pouvez même gagner encore plus de vitesse en chargeant les données dans une table avec de nombreux index en utilisant la procédure suivante :
    1. Créez une table avec `mysql` ou Perl avec `CREATE TABLE`.
    2. Exécutez `mysqladmin flush-tables`.
    3. Utilisez `isamchk --keys-used=0 -rq /path/to/db/nom_table`. Cela va effacer tous les index de cette table.
    4. Insérez les données dans la table avec `LOAD DATA INFILE`.
    5. Si vous avez `pack_isam` et voulez compresser les tables, lancez `pack_isam`.
    6. Recréez les index avec `isamchk -r -q /path/to/db/nom_table`.
    7. Exécutez `mysqladmin flush-tables`.
- Pour gagner un peu de vitesse avec `LOAD DATA INFILE` et `INSERT`, agrandissez le buffer de clés. Cela peut se faire avec l'option `-O key_buffer=#` option de `mysqld` ou `safe_mysqld`. Par exemple, 16M est une bonne valeur si vous disposez de beaucoup de RAM
- Lorsque vous créez des textes d'exportation pour les réutilisez dans d'autres programmes, utilisez `SELECT ... INTO OUTFILE`. Reportez vous à la section `LOAD DATA`.
- Lorsque vous faites des insertions ou des modifications nombreuses et successives, vous pouvez gagner de la vitesse en verrouillant la table avec `LOCK TABLES`. `LOAD DATA INFILE` et `SELECT ...INTO OUTFILE` sont atomique, ce qui rend inutile l'utilisation de `LOCK TABLES` lors de leur utilisation. `LOCK TABLES`.

Pour vérifier l'état de fragmentation de vos tables, utilisez `isamchk -evi` sur le fichier``.ISM''. Reportez vous à la section 13 Maintenance d'un serveur MySQL

10.13 Problèmes liés au verrouillage

Le système de verrouillage des tables de ***MySQL*** empêche les blocages.

MySQL utilise le verrouillage de table pour réaliser des verrous à très haute vitesse. Cela remplace avantageusement le verrouillage de colonne ou de ligne. Pour les grandes tables, le verrouillage est BEAUCOUP plus rapide que le verrouillage de ligne. Cependant, il y a quelques limitations.

Le verrouillage de table permet à plusieurs threads de lire une table en même temps, mais limite l'écriture de

cette table à un seul thread, qui doit s'assurer de l'accès exclusif à la table. Durant la modification, les autres threads devront attendre.

Etant donné que les modifications sont considérés comme plus importantes que les SELECT, toutes les commandes de modifications ont une priorité supérieure aux commandes de selections. Cela permet de ne pas bloquer les modifications si une table est sujette à un fort trafic en lecture.

Le problème principal avec ceci est décrit avec la situation suivante :

- Un client émet une requête SELECT qui prend un très long temps.
- Un autre client émet une requête INSERT sur la même table; ce client va devoir attendre que le premier a fini.
- Un autre client émet une autre requête SELECT sur la même table; Comme INSERT a la priorité sur SELECT, ce SELECT va attendre que INSERT soit fini. Il va aussi attendre que le premier SELECT soit fini !

Les solutions envisageables sont :

- Essayer d'accélérer la commande SELECT; comme par exemple, en créant une table sommaire.
- Démarrer mysqld avec --low-priority-inserts. Cela va donner aux commandes de sélection la priorité sur les commandes d'insertion. Dans ce cas, la dernière requête SELECT du scénario passera en priorité sur la commande d'insertion.
- Vous pouvez donnez spécifiquement à une commande INSERT,UPDATE ou DELETE une priorité inférieure avec l'attribut LOW_PRIORITY.
- Vous pouvez spécifiez que toutes les modifications d'un thread donnée auront une priorité basse, en utilisant la commande SQL : : SET SQL_LOW_PRIORITY_UPDATES=1. Reportez vous à la section SET OPTION.
- Vous pouvez donnez spécifiquement à une commande SELECT une priorité supérieure avec l'attribut HIGH_PRIORITY. Reportez vous à la section SELECT.
- Si vous utilisez fréquemment des commandes INSERT et SELECT, l'attribut DELAYED de INSERT pourra vous aider à résoudre votre problème. Reportez vous à la section INSERT.
- Si vous avez des problèmes avec SELECT et DELETE, l'option LIMIT de DELETE peut être utile. Reportez vous à la section DELETE.

10.14 Facteur affectant la vitesse des INSERT

Le temps nécessaire pour insérer une ligne comprend :

- Connexions: (3)
- Envoi de la requête au serveur: (2)
- Analyse la requête: (2)
- Insertion de la ligne record: (1 x size of record)
- Insertion indexes: (1 x indexes)
- Terminaison: (1)

Le (nombre) est proportonnel au temps.. Le détail ci−dessus ne prend pas en compte le temps d'ouverture initial des tables (qui sera fait à chaque nouvel accès d'une requête).

Ma taille d'une table ralenti l'insertion des index au rythme de N log N (B−tree).

Vous pouvez accélèrer les insertions en verrouillant votre table et/ou en utilisant des insertions multiples. Les insertions multiples peuvent être jusqu'à 5 fois plus rapides que des insertions séparées :

```
mysql62; LOCK TABLES a WRITE;
mysql62; INSERT INTO a VALUES (1,23),(2,34),(4,33);
mysql62; INSERT INTO a VALUES (8,26),(6,29);
mysql62; UNLOCK TABLES;
```

La principale différence vient du fait que le buffer d'index est écrit sur le disque une seule fois, après toutes les insertions. Dans l'autre cas, il y aurait de nombreuses écritures sur le disque, une par commande INSERT. Le verrouillage n'est pas nécessaire si vous pouvez insérer toutes les lignes dans une seule commande d'insertion.

Le verrouillage va aussi réduire le nombre de test de connexions multiples, mais il va aussi augmenter le temps d'attente des autres threads. Par exemple :

```
thread 1 does 1000 inserts
thread 2, 3, and 4 does 1 insert
thread 5 does 1000 inserts
```

Si vous n'utilisez pas le verrouillage, , 2, 3 et 4 vont s'achever avant 1 et 5. Si vous utilisez le verrouillage, 2, 3 et 4 ne vont probablement pas finir avant 1 ou5, mais le temps total devrait être réduit de 40%.

Etant donné que les commandes `INSERT`, `UPDATE` et `DELETE` sont extrêmement rapide sous ***MySQL***, vous obtiendrez une performance générale accrue si vous utilisez le verrouillage dès que vous mettez en jeu plus de 5 commandes. Vous pouvez aussi déverrouiller et reverrouiller une table de temps à autre (si vous inserez énormément de lignes, par exemple), pour donner un peu de temps aux autres threads. Cela peut fournir un gain significatif de performances.

Bien entendu, `LOAD DATA INFILE` est toujours le plus efficace.

Si vous insérez de nombreuses lignes de différents clients, vous pouvez accélérer la procédure en utilisant l'option `INSERT DELAYED`. `INSERT.`

10.15 Facteur affectant la vitesse des `DELETE`

Le temps d'effacement est exactement proportionnel au nombre d'index. Pour effacer des lignes plus rapidement, vous pouvez augmenter la taille du cache d'index. La valeur par défaut est 1M; et pour accélérer MySQL, vous devez l'augmenter de plusieurs fois cette valeur(Essayez 16M si vous avez assez de mémoire)

10.16 Comment faire tourner MySQL à pleine vitesse?

Commencez par tester votre installation ! Vous pouvez prendre n'importe quel programme de la suite de test ***MySQL*** (normalement fournie dans le dossier ``` `sql−bench' ```) et la modifier selon vos besoin. Vous pourrez ainsi essayer différentes solutions, et trouver celle qui est le plus rapide :

- Démarrez `mysqld` avec les options adéquates. Plus de mémoire accélère MySQL, si vous en avez.. 10.1 Optimisation des valeurs du serveur.
- Créez des index pour rendre vos sélections rapides. 10.4 Utilisation des index.
- Optimisez vos types de colonnes pour qu'ils soient aussi efficient que possible. Par exemple, déclarez les colonnes `NOT NULL` si possible. 10.12 Comment constituer une table pour qu'elle soit petite et rapide?.
- MySQL a deux niveaux de verrouillage : un verrouillage interne, un verrouillage externe. Le verrouillage interne s'assure que les modification / lecture d'informations sont atomiques et ne s'exécutent pas de manière concurente. Le verrouillage externe permet d'avoir plusieurs serveurs ***MySQL*** qyu utilisent les mêmes données, mais aussi d'utiliser `isamchk` pour vérifier les tables sans arrêter le serveur `mysqld`. L'option `−−skip−locking` désactive le verrouillage externe entre deux requête SQL. Cela accélère les traitements mais il y a des inconvénients :
  - Vous devez ABSOLUMENT exécuter `mysqladmin flush−tables` avec de vérifier et réparer des tables avec `isamchk`. (`isamchk −d nom_table` est toujours autorisé, car c'est un simple affichage d'informations).
  - Il est impossible d'exécuter deux serveurs ***MySQL*** sur les mêmes données, car il y a un risque d'accès concurent.

L'option `−−skip−locking` est mise par défaut lors de la compilation avec MIT−pthreads, car `flock()`n'est pas supporté par MIT−pthreads sur toutes les plate−formes. Le seul ca où il est impossible d'utiliser `−−skip−locking` est lorsque vous avez plusieurs SERVEUR ***MySQL*** (pas les clients) avec les mêmes données (ou bien exécutez `isamchk` sur une table sans avoir vidé les tables de la mémoire). Vous pouvez toujours utiliser use `LOCK TABLES` / `UNLOCK TABLES` même si vous avec mis l'option −−skip−locking

- Si les modifications posent des problèmes, vous pouvez les reportez puis les regrouper pour une exécution ultérieure. Les modifications multiples sont beaucoup plus rapides que des modifications uniques.
- Sous FreeBSD, si le problème vient des MIT−pthreads, la mise à jour vers la version FreeBSD 3.0 (ou plus récent) devrait résoudre les problèmes.. Cette mise à jour rend possible l'utilisation des sockets Unix (avec FreeBSD, c'est beaucoup plus rapide que TCP/IP avec MIT−pthreads) et le package de threads est bien mieux intégré.
- La vérification de droits sur une table au niveau des colonnes dégrade sérieusement les performances.

Si votre problème porte sur une fonction particulière de ***MySQL***, vous pouvez toujours la chronométrer avec le client ***MySQL***:

```
mysql62; select benchmark(1000000,1+1);
+------------------------+
| benchmark(1000000,1+1) |
+------------------------+
|                      0 |
+------------------------+
1 row in set (0.32 sec)
```

L'exemple ci dessus montre que ***MySQL*** peut exécuter plus de 1,000,000 de fois l'expressions en 0.32 seconde sur un simple `PentiumII 400MHz`.

Toutes les fonctions ***MySQL*** sont très optimisées ; mais il peut y avoir des exceptions, et l'utilitaire `benchmark(loop_count,expression)` est un bon outil pour étudier votre problème.

10.17 Quel sont les différents formats de lignes? Quand utiliser VARCHAR/CHAR?

MySQL n'a pas de type SQL VARCHAR à proprement parler.

A la place, , ***MySQL*** possède trois manières d'enregistrer les informations, et d'émuler le type VARCHAR.

Si une table n'a aucune colonne de type VARCHAR, BLOB ou TEXT, la table est considéré à taille de ligne constante. Sinon, c'est une ligne à taille variable. Les colonnes CHAR et VARCHAR sont alors traitées de manière identique du point de vue de l'application : dans les deux cas, les espaces de fin de chaînes sont supprimés à la lecture.

Vous pouvez vérifier le format utilisé dans une table avec `isamchk -d` (`-d` signifie ``description de la table``).

MySQL possède trois formats de tables différents : taille fixe, taille variable, compressé. En voici la comparaison :

Tables à taille fixe

- C'est le format par défaut. Il est utilisé lorsqu'une table ne contient pas de colonne de type VARCHAR, BLOB ou TEXT.
- Toutes les colonnes de type CHAR, NUMERIC et DECIMAL sont complétée par des espaces pour remplir la colonne
- Tres rapide
- Facile à mettre en cache
- Facile à reconstruire après un crash, car tous les enregistrements ont une position fixe.
- N'a pas besoin de réorganisation (avec `isamchk`) à moins q'un nombre important d'enregistrement ne soit effacé, et que vous vouliez libérer de l'espace disque pour le serveur.
- Généralement, requiert plus de place que les tables à taille variable

Tables à taille variable

- Ce format est utilisé pour gérer le tables qui contiennent des colonnes de type VARCHAR, BLOB ou TEXT.
- Toutes les colonnes de type chaîne sont de taille variable (à l'exception de celle qui font moins de 4 caractères)
- Car enregistrement est précédé d'un octet qui indique quelle colonnes sont vides (''), ou nulle (suivant le type de la colonne). (Ne pas confondre avec les valeurs NULL). Si une colonne de type chaîne a une longueur de 0 après suppression des espaces de remplissage, ou bien si une colonne de type numérique à une valeur de 0, cette information est reportée dans cet octet, et la valeur n'est pas sauvée. Les chaînes non vides ont une longueur d'un octet de plus que le nombre de caractères qu'elles contiennent.
- En général, ces tables prennent moins de place sur le disque.
- Chaque enregistrement utilise uniquement l'espace dont il a besoin. Si le record augmente en taille, il est scindé en plusieurs parties. Cela aboutit à la fragmentation de la table.
- Si vous mettez à jour un enregistrement qui a besoin de plus de place, l'enregistrement sera fragmenté. Dans ce cas, il vous faudra exécuter `isamchk −r` de temps en temps pour améliorer les performances d'ensemble. Utilisez `isamchk −ei nom_table` pour avoir des informations concernant la fragmentation d'une table.
- Ce type de table n'est pas simple à reconstruire après un crash, car les enregistrements peuvent être fragmenté.
- On peut s'attendre à ce qu'un enregistrement prennent la place suivante :

```
3

+ (nombre de colonnes + 7) / 8

+ (nombre de colonne de type char)

+ taille compactee des colonnes de type numerique

+ longueur des chaînes

+ (nombre de valeur NULL + 7) / 8
```

Il y a aussi une pénalité de 6 octet pour chaque fragment de record. Un record de taille variable est fragmenté à chaque fois qu'il grandit de taille. Chaque nouveau fragment prendra au moins 20 octets, ce qui fait que plusieurs aggrandissements pourront partager le même fragment. Sinon, un autre fragment sera généré. Vous pouvez compter le nombre de fragment avec la commande `isamchk −ed`. Tous ces fragments peuvent être supprimés avec `isamchk −r`.

Table compressée

- C'est une table accessible uniquement en lecture, et constituée gr ce à `pack_isam`. Tous les clients avec une garantie étendue par email ont droit à une copie de l'utilitaire `pack_isam` pour leur utilisation personnelle.
- L'utilitaire de décompression est fourni avec toutes les distributions de ***MySQL*** ce qui fait que tout le monde peut accéder aux tables comrpessées, par `pack_isam` (ATTENTION : la comrpession est plate forme dépendante.)
- Prend très peu de place sur le disque. Réduit l'espace disque au minimum.
- Chaque enregistrement est compressé séparément. Les entêtes d'un enregistrement sont de taille fixe (1−3octets) suivant la taille du plus grand enregistrement de la table. Chaque colonne est compressée différemment. Voici les types de compression :
  - Compression de type Huffman, différente pour chaque colonne de la table.
  - Compression des suffixes
  - Compression des prefixes
  - Les valeurs nulles (`0`) sont stockés sur un bit.
  - Les valeurs entières sont stockées au format le plus petit possible. Par exemple, un `BIGINT` (8 octets) peut être stocké au format `TINYINT` (1 octet) si sa valeur est entre `0` et `255`.
  - Si une colonne a un petit nombre de valeur différente, elle est converti en `ENUM`.
  - Une colonne peut utiliser une combinaison de toutes les techniques ci dessus.
- Gère les tables de taille fixe ou variable, mais pas les types `BLOB` ou `TEXT`.
- Peut être décompressé par `isamchk`.

MySQL peut supporter différents types d'index, mais le format standard est NISAM. C'est un index de type B−tree dont on peut grossièrement calculer la taille avec la formule `(key_length+4)*0.67`, appliquée à toutes les clés. (Ce qui correspond au pire cas, quand toutes les clés sont insérée en ordre).

Les index de chaînes subissent une compression sur les espaces. Si la première partie de l'index est une chaîne, elle sera aussi compressée. La compression des espaces réduit la taille si la colonne a beaucoup d'espace de remplissage, ou si une colonne de type VARCHAR n'est pas toujours remplie. La compression par préfixe est particulièrement utile si plusieurs chaînes commencent de même.

10.18 Types de tables MySQL

Les types de table ont été introduit à partir de ***MySQL*** 3.23!

Lors de la création d'une table, vous pouvez préciser à ***MySQL*** quel type de table utiliser. ***MySQL*** va, dans tous les cas, créer un fichier `.frm` pour y noter la structure de la table, et les définitions des colonnes. Suivant le type de table, l'index et les données seront enregistrées dans d'autres fichiers.

Vous pouvez convertir des tables d'un type à l'autre avec la commande `ALTER TABLE`. Reportez vous à la section `ALTER TABLE`.

- `ISAM` Ce type est le type original de pour les tables ***MySQL***. Il utilise un index `B−tree`. L'index est enregistré dans un fichier avec l'extension `.ISM` et les données sont enregistrées dans un fichier avec l'extension `.ISD`. Vous pouvez vérifier et réparer les tables `ISAM` avec l'utilitaire `isamchk`. Reportez vous à la section 13.4 Utiliser `isamchk` pour réparer une table. Les tables `ISAM` ne are sont pas portables d'une plate forme à l'autre. Les fichiers `ISAM` ont les caractéristiques suivantes :
  - Clés compressées et de longueur fixe
  - Enregistrement de longueur fixe ou variable
  - 16 clés avec 16 parties de clé par clé
  - Longueur maximale d'une clé : 256 (par défaut)
  - Les informations sont enregistrées dans un format propre à la machine. Rapide mais pas portable.
- `MyISAM` est le type de table par défaut à partir de ***MySQL*** 3.23. Il est basé sur le code de `ISAM` et dispose de nombreuses amélioration : L'index est desormais enregistré dans le fichier `.MYI` et les données sont enregistrées dans le fichier `.MYD`. Vous pouvez vérifier et réparer des fichiers `MyISAM` avec l'utilitaire `myisamchk`. 13.4 Utiliser `isamchk` pour réparer une table. Les caractéristiques suivantes sont nouvelles:
  - Support de gros fichiers (63 bits) sur les OS qui le supporte.
  - Les données sont toutes sauvegardées avec le bit de poids faible en premier. `MyISAM` est donc indépendant de la machine et de l'OS.
  - Les clés sont enregistrées avec le bit de poids fort en premier, pour améliorer la compression.
  - Gestion interne des colonnes `AUTO_INCREMENT`. `MyISAM` va automatiquement mettre à jour ces colonnes lors des commandes `INSERT/UPDATE`. Les commandes `AUTO_INCREMENT` peuvent être modifiée avec `myisamchk`. Tout ceci accélère les traitements sur les colonnes `AUTO_INCREMENT` et les anciens numéros ne seront par réutilisé.
  - Taille maximale des clés : 500 par défaut. Dans le cas ou les clés dépassent la taille de 250, un bloc de clé plus grand est généré et utilisé.
  - Nombre maximum de clés par table est porté à 32 par défaut. Il peut même être étendu à 64 sans avoir à recompiler `myisamchk`.
  - Un flag dans `MyISAM` indique si le fichier a bien été fermé ou non. Cela permettra bientôt de faire des réparation automatiques.
  - `myisamchk` marque les tables.. `myisamchk −−fast` ne vérifiera que les talbes qui ne sont pas marquées.
  - `myisamchk −a` crée des statistiques sur les parties de clés (et non plus la totalité des clés, comme avec `NISAM`).
  - Les enregistrements de taille dynamiques sont bien moins fragmentés lors de modifications et effacements. Les blocs effacés et adjacents sont combinés pour agrandir les enregistrements.
  - `myisampack` (appelé `pack_isam` sous `NISAM`) comrpess les colonnes `BLOB` et VARCHAR

MyISAM supporte les caractéristiques suivantes, que ***MySQL*** sera bientôt capable d'exploiter :

  - Les colonnes `BLOB` et `TEXT` peuvent être indexées/
  - Support d'un véritable type `VARCHAR` : Les colonnes de type `VARCHAR` commence par une longueur enregistré sur 2octets.
  - Les tables avec des colonnes de type `VARCHAR` peut avoir des enregistrements de longueur variable ou fixes.
  - `VARCHAR` et `CHAR` peuvent atteindre la taille de 64Ko. Tous les segments de clés on leur propres définitions. Cela permettra à ***MySQL*** d'utiliser des langages différents pour chaque colonne.
  - `NULL` sont autorisés dans les colonnes indexées. Cela prend de 0 à 1 octet par clé.
  - Un index hashé peut être utilisé comme `UNIQUE`; Cela va permettre d'avoir l'attribut `UNIQUE` pour n'importe quelle combinaison de colonne d'une table. (On ne pourra pas faire de recherche sur ces index).
- `HEAP` Ces tables utilisent un index hashé est sont enregistrées en mémoire. Cela les rend beaucoup plus rapide, mais si ***MySQL*** crashe, toutes les données seront perdues. `HEAP` est un bon type pour les tables temporaire !

```
CREATE TABLE test TYPE=HEAP SELECT ip,SUM(downloads) as down FROM log_table GROUP BY ip;

SELECT COUNT(ip),AVG(down) from test;

drop table test;
```

Voici quelques considérations sur les tables `HEAP`:

- Vous devriez toujours spécifier le nombre maximal de ligne (`MAX_ROWS`) dans la commande `CREATE` pour s'assurer que la table ne prend pas toute la mémoire.
- Les index ne seront utilisé qu'avec les opérateurs = et <=> (mais TRES rapide)
- `HEAP` est une table à enregistrement de taille fixe.
- `HEAP` n'accepte pas les types `BLOB/TEXT`.
- HEAP n'accepte pas les colonnes `AUTO_INCREMENT`.
- HEAP n'accepte pas les index sur les colonnes avec des valeurs `NULL`.
- Vous pouvez avoir des clés non unique dans une table `HEAP` (ce qui n'est pas le cas des tables hashées normales)
- Les tables `HEAP` sont partagées entre tous les clients (comme n'importe quelle table)
- Les données des tables `HEAP` sont allouées par petits blocs. Ces tables sont purement dynamiques en insertion. Pas de dépassement de capacité, ni d'espace de clé supplémentaires à prévoir. Les lignes effacées sont placées dans une liste, et seront réutilisées à la prochaine insertion.
- Pour libérer de la mémoire, vous devez exécuter la commande `DELETE FROM heap_table` ou `DROP TABLE heap_table`.
- Pour s'assurer que vous ne faite rien de grave, la taille des tables `HEAP` est limitée à `max_heap_table_size`.

11 La suite de tests de MySQL

This should contain a technical description of the ***MySQL*** benchmark suite (and `crash-me`) but that description is not written yet. Currently, you should look at the code and results in the `` `bench' `` directory in the distribution (and of course on the web page at http://www.mysql.com/crash-me-choose.htmy).

It is meant to be a benchmark that will tell any user what things a given SQL implementation performs well or poorly at.

`crash-me` tries to determine what features a database supports and what its capabilities and limitations are by actually running queries. For example, it determines:

- What column types are supported
- How many indexes are supported
- What functions are supported
- How big a query can be
- How big a VARCHAR column can be

12 Utilitaires MySQL

12.1 Présentation des différents programmes MySQL

All ***MySQL*** clients that communicate with the server using the `mysqlclient` library use the following environment variables:

Name	*Description*
`MYSQL_UNIX_PORT`	The default socket; used for connections to `localhost`
`MYSQL_TCP_PORT`	The default TCP/IP port
`MYSQL_PWD`	The default password
`MYSQL_DEBUG`	Debug−trace options when debugging
`TMPDIR`	The directory where temporary tables/files are created

Use of `MYSQL_PWD` is insecure. 6.3 Connection au serveur MySQL.

The `` `mysql' `` client uses the file named in the `MYSQL_HISTFILE` environment variable to save the command line history. The default value for the history file is `` `$HOME/.mysql_history' ``, where `$HOME` is the value of the `HOME` environment variable.

All ***MySQL*** programs take many different options. However, every ***MySQL*** program provides a `--help` option that you can use to get a full description of the program's different options. For example, try `mysql --help`.

You can override default options for all standard client programs with an option file. 4.15.4 Fichier d'options.

The list below briefly describes the ***MySQL*** programs:

`isamchk`
: Utility to describe, check, optimize and repair ***MySQL*** tables. Because `isamchk` has many functions, it is described in its own chapter. 13 Maintenance d'un serveur MySQL.

`make_binary_release`
: Makes a binary release of a compiled ***MySQL***. This could be sent by FTP to `` `/pub/mysql/Incoming' `` on `ftp.tcx.se` for the convenience of other ***MySQL*** users.

`msql2mysql`
: A shell script that converts `mSQL` programs to ***MySQL***. It doesn't handle all cases, but it gives a good start when converting.

`mysql`
: `mysql` is a simple SQL shell (with GNU `readline` capabilities). It supports interactive and non−interactive use. When used interactively, query results are presented in an ASCII−table format. When used non−interactively (e.g., as a filter), the result is presented in tab−separated format. (The output format can be changed using command−line options.) You can run scripts simply like this:

```
shell62; mysql database 60; script.sql 62; output.tab
```

If you have problems due to insufficient memory in the client, use the `--quick` option! This forces `mysql` to use `mysql_use_result()` rather than `mysql_store_result()` to retrieve the result set.

`mysqlaccess`
: A script that checks the access privileges for a host, user and database combination.

`mysqladmin`
: Utility for performing administrative operations, such as creating or dropping databases, reloading the grant tables, flushing tables to disk and reopening log files. `mysqladmin` can also be used to retrieve version, process and status information from the server. `mysqladmin.`

`mysqlbug`
: The ***MySQL*** bug report script. This script should always be used when filing a bug report to the ***MySQL*** list.

`mysqld`

The SQL daemon. This should always be running.

mysqldump
: Dumps a ***MySQL*** database into a file as SQL statements or as tab−separated text files. Enhanced freeware originally by Igor Romanenko. mysqldump.

mysqlimport
: Imports text files into their respective tables using `LOAD DATA INFILE`. mysqlimport.

mysqlshow
: Displays information about databases, tables, columns and indexes.

mysql_install_db
: Creates the ***MySQL*** grant tables with default privileges. This is usually executed only once, when first installing ***MySQL*** on a system.

replace
: A utility program that is used by `msql2mysql`, but that has more general applicability as well. `replace` changes strings in place in files or on the standard input. Uses a finite state machine to match longer strings first. Can be used to swap strings. For example, this command swaps `a` and `b` in the given files:

```
shell62; replace a b b a −− file1 file2 ...
```

safe_mysqld
: A script that starts the `mysqld` daemon with some safety features, such as restarting the server when an error occurs and logging runtime information to a log file.

12.2 Administrer un serveur MySQL

Utility for performing administrative operations. The syntax is:

```
shell62; mysqladmin [OPTIONS] command [command−option] command ...
```

You can get a list of the options your version of `mysqladmin` supports by executing `mysqladmin −−help`.

The current `mysqladmin` supports the following commands:

create databasename	Create a new database.
drop databasename	Delete a database and all its tables.
extended−status	Gives an extended status message from the server.
flush−hosts	Flush all cached hosts.
flush−logs	Flush all logs.
flush−tables	Flush all tables.
flush−privileges	Reload grant tables (same as reload)
kill id,id,...	Kill mysql threads.
password	new−password Change old password to new−password
ping	Check if mysqld is alive
processlist	Show list of active threads in server
reload	Reload grant tables
refresh	Flush all tables and close and open logfiles
shutdown	Take server down
status	Gives a short status message from the server
variables	Prints variables available
version	Get version info from server

All commands can be shortened to their unique prefix. For example:

```
shell62; mysqladmin proc stat
+----+-------+-----------+----+-------------+------+-------+------+
| Id | User  | Host      | db | Command     | Time | State | Info |
+----+-------+-----------+----+-------------+------+-------+------+
| 6  | monty | localhost |    | Processlist | 0    |       |      |
+----+-------+-----------+----+-------------+------+-------+------+
Uptime: 10077  Threads: 1  Questions: 9  Slow queries: 0  Opens: 6  Flush tables: 1
Open tables: 2  Memory in use: 1092K  Max memory used: 1116K
```

The `mysqladmin status` command result has the following columns:

Uptime	Number of seconds the ***MySQL*** server have been up
Threads	Number of active threads (clients)
Questions	Number of questions from clients since `mysqld` was started
Slow requêtes@tab Requêtesthat has taken more than `long_query_time` seconds	
Opens	How many tables `mysqld` has opened.
Flush tables	Number of `flush ...`, `refresh` and `reload` commands.
Open tables	Number of tables that are open now
Memory in use	Memory allocated directly by the mysqld code (only available when ***MySQL*** is compiled with −−with−debug)
Max memory used	Maximum memory allocated directly by the mysqld code (only available when ***MySQL*** is compiled with −−with−debug)

12.3 Sauver la structure et les données des bases et tables MySQL

Utility to dump a database or a collection of database for backup or for transferring the data to another SQL server. The dump will contain SQL statements to create the table and/or populate the table.

```
shell62; mysqldump [OPTIONS] database [tables]
```

If you don't give any tables, the whole database will be dumped.

You can get a list of the options your version of `mysqldump` supports by executing `mysqldump −−help`.

Note that if you run `mysqldump` without `−−quick` or `−−opt`, `mysqldump` will load the whole result set into memory before dumping the result. This will probably be a problem if you are dumping a big database.

`mysqldump` supports the following options:

`−−add−locks`
: Add `LOCK TABLES` before and `UNLOCK TABLE` after each table dump. (To get faster inserts into ***MySQL***).

`−−add−drop−table`
: Add a `drop table` before each create statement.

`−−allow−keywords`
: Allow creation of column names that are keywords. This works by prefixing each column name with the table name.

`−c, −−complete−insert`
: Use complete insert statements (with column names).

`−C, −−compress`
: Compress all information between the client and the server if both support compression.

`−−delayed`
: Insert rows with the `INSERT DELAYED` command.

`−e, −−extended−insert`

Use the new multiline `INSERT` syntax. (Gives more compact and faster inserts statements)

−#, −−debug[=option_string]
Trace usage of the program (for debugging).

−−help
Display a help message and exit.

−−fields−terminated−by=...
−−fields−enclosed−by=...
−−fields−optionally−enclosed−by=...
−−fields−escaped−by=...
−−fields−terminated−by=...
These options are used with the `−T` option and have the same meaning as the corresponding clauses for `LOAD DATA INFILE`. LOAD DATA.

−F, −−flush−logs
Flush logs file in the ***MySQL*** server before starting the dump.

−f, −−force,
Continue even if we get an SQL error during a table dump.

−h, −−host=..
Dump data from the ***MySQL*** server on the named host. The default host is `localhost`.

−l, −−lock−tables.
Lock all tables for starting the dump.

−t, −−no−create−info
Don't write table creation info (The `CREATE TABLE` statment)

−d, −−no−data
Don't write any row information for the table. This is very useful if you just want to get a dump of the structure for a table!

−−opt
Same as `−−quick −−add−drop−table −−add−locks −−extended−insert −−use−locks`. Should give you the fastest possible dump for reading into a ***MySQL*** server.

−pyour_pass, −−password[=your_pass]
The password to use when connecting to the server. If you specify no `` `=your_pass' `` part, `mysqldump` solicits the password from the terminal.

−P port_num, −−port=port_num
The TCP/IP port number to use for connecting to a host. (This is used for connections to hosts other than `localhost`, for which Unix sockets are used.)

−q, −−quick
Don't buffer query, dump directly to stdout; Uses `mysql_use_result()` to do this.

−S /path/to/socket, −−socket=/path/to/socket
The socket file to use when connecting to `localhost` (which is the default host).

−T, −−tab=path−to−some−directory
Creates a `table_name.sql` file, that conntains the SQL CREATE commands, and a `table_name.txt` file, that contains the data, for each give table. ***NOTE***: This only works if `mysqldump` is run on the same machine as the `mysqld` daemon. The format of the `.txt` file is made according to the `−−fields−xxx` and `−−lines−−xxx` options.

−u user_name, −−user=user_name
The ***MySQL*** user name to use when connecting to the server. The default value is your Unix login name.

−O var=option, −−set−variable var=option
Set the value of a variable. The possible variables are listed below.

−v, −−verbose
Verbose mode. Print out more information what the program does.

−V, −−version
Print version information and exit.

−w, −−where='where−condition'
Dump only selected records; Note that QUOTES are mandatory!
`"−−where=user='jimf'" "−wuserid62;1" "−wuserid60;1"`

The most normal use of `mysqldump` is probably for making a backup of whole database:

```
mysqldump −−opt database 62; backup−file.sql
```

But it's also very useful to populate another ***MySQL*** server with information from a databas:

```
mysqldump −−opt database | mysql −−host=remote−host −C database
```

12.4 Importer des données depuis un fichier texte

`mysqlimport` provides a command line interface to the `LOAD DATA INFILE` SQL statement. Most options to `mysqlimport` correspond directly to the same options to `LOAD DATA INFILE`. LOAD DATA.

`mysqlimport` is invoked like this:

```
shell62; mysqlimport [options] filename ...
```

For each text file named on the command line, `mysqlimport` strips any extension from the filename and uses the result to determine which table to import the file's contents into. For example, files named `` `patient.txt' ``, `` `patient.text' `` and `` `patient' `` would all be imported into a table named `patient`.

`mysqlimport` supports the following options:

−C, −−compress
: Compress all information between the client and the server if both support compression.

−#, −−debug[=option_string]
: Trace usage of the program (for debugging).

−d, −−delete
: Empty the table before importing the text file.

−−fields−terminated−by=...
−−fields−enclosed−by=...
−−fields−optionally−enclosed−by=...
−−fields−escaped−by=...
−−fields−terminated−by=...
: These options have the same meaning as the corresponding clauses for `LOAD DATA INFILE`. LOAD DATA.

−f, −−force
: Ignore errors. For example, if a table for a text file doesn't exist, continue processing any remaining files. Without `−−force`, `mysqlimport` exits if a table doesn't exist.

−−help
: Display a help message and exit.

−h host_name, −−host=host_name
: Import data to the ***MySQL*** server on the named host. The default host is `localhost`.

−i, −−ignore
: See the description for the `−−replace` option.

−l, −−lock−tables
: Lock ***ALL*** tables for writing before processing any text files. This ensures that all tables are synchronized on the server.

−L, −−local
: Read input files from the client. By default, text files are assumed to be on the server if you connect to `localhost` (which is the default host).

−pyour_pass, −−password[=your_pass]
: The password to use when connecting to the server. If you specify no `` `=your_pass' `` part, `mysqlimport` solicits the password from the terminal.

−P port_num, −−port=port_num
: The TCP/IP port number to use for connecting to a host. (This is used for connections to hosts other than `localhost`, for which Unix sockets are used.)

−r, −−replace
: The `−−replace` and `−−ignore` options control handling of input records that duplicate existing records on unique key values. If you specify `−−replace`, new rows replace existing rows that have the same unique key value. If you specify `−−ignore`, input rows that duplicate an existing row on a unique key value are skipped. If you don't specify either option, an error occurs when a duplicate key value is found, and the rest of the text file is ignored.

−s, −−silent
: Silent mode. Write output only when errors occur.

−S /path/to/socket, −−socket=/path/to/socket
: The socket file to use when connecting to `localhost` (which is the default host).

−u user_name, −−user=user_name
: The ***MySQL*** user name to use when connecting to the server. The default value is your Unix login name.

−v, −−verbose
: Verbose mode. Print out more information what the program does.

−V, −−version
: Print version information and exit.

12.5 Le générateur de tables compressée

`pack_isam` is an extra utility that you get when you order more than 10 licenses or extended support. Since

pack_isam is distributed only in binary form, pack_isam is available only on some platforms.

Of course, all future updates to pack_isam are included in the price. pack_isam may at some time be included as standard when we get some kind of turnover for ***MySQL***.

pack_isam works by compressing each column in the table separately. The information needed to decompress columns is read into memory when the table is opened. This results in much better performance when accessing individual records, since you only have to uncompress exactly one record, not a much larger disk block like when using Stacker on MS−DOS. Usually, pack_isam packs the data file 40%−70%.

MySQL uses memory mapping (mmap()) on compressed tables and falls back to normal read/write file usage if mmap() doesn't work.

There are currently two limitations with pack_isam:

- After packing, the table is read only.
- It can't pack BLOB columns, yet.

Fixing these limitations is on our TODO list but with low priority.

pack_isam is invoked like this:

```
shell62; pack_isam [options] filename ...
```

Each filename should be the name of an index (`.ISM') file. If you are not in the database directory, you should specify the pathname to the file. It is permissible to omit the `.ISM' extension.

pack_isam supports the following options:

−b, −−backup
: Make a backup of the table as nom_table.OLD.

−#, −−debug=debug_options
: Output debug log. The debug_options string often is 'd:t:o,filename'.

−f, −−force
: Force packing of the table even if it becomes bigger or if the temporary file exists. (pack_isam creates a temporary file named `nom_table.TMD' while it compresses the table. If you kill pack_isam, the `.TMD' file may not be deleted. Normally, pack_isam exits with an error if it finds that `nom_table.TMD' exists. With −−force, pack_isam packs the table anyway.

−?, −−help
: Display a help message and exit.

−j big_nom_table, −−join=big_nom_table
: Join all tables named on the command line into a single table big_nom_table. All tables that are to be combined MUST be identical (same column names and types, same indexes, etc.)

−p #, −−packlength=#
: Specify the record length storage size, in bytes. The value should be 1, 2 or 3. (pack_isam stores all rows with length pointers of 1, 2 or 3 bytes. In most normal cases, pack_isam can determine the right length value before it begins packing the file, but it may notice during the packing process that it could have used a shorter length. In this case, pack_isam will print a note that the next time you pack the same file, you could use a shorter record length.)

−s, −−silent
: Silent mode. Write output only when errors occur.

−t, −−test
: Don't pack table, only test packing it.

−T dir_name, −−tmp_dir=dir_name
: Use the named directory as the location in which to write the temporary table.

−v, −−verbose
: Verbose mode. Write info about progress and packing result.

−V, −−version
: Display version information and exit.

−w, −−wait
: Wait and retry if table is in use. If the mysqld server was invoked with the −−skip−locking option, it is not a good idea to invoke pack_isam if the table might be updated during the packing process.

The sequence of commands shown below illustrates a typical table compression session:

```
shell62; ls −l station.*
−rw−rw−r−−   1 monty    my         994128 Apr 17 19:00 station.ISD
−rw−rw−r−−   1 monty    my          53248 Apr 17 19:00 station.ISM
−rw−rw−r−−   1 monty    my           5767 Apr 17 19:00 station.frm

shell62; isamchk −dvv station

ISAM file:     station
Isam−version:  2
Creation time: 1996−03−13 10:08:58
Recover time:  1997−02−02  3:06:43
Data records:              1192  Deleted blocks:              0
Datafile: Parts:           1192  Deleted data:                0
Datafile pointer (bytes):     2  Keyfile pointer (bytes):     2
Max datafile length:   54657023  Max keyfile length:   33554431
Recordlength:               834
Record format: Fixed length

table description:
Key Start Len Index   Type                        Root  Blocksize    Rec/key
1   2     4   unique  unsigned long               1024       1024          1
2   32    30  multip. text                       10240       1024          1

Field Start Length Type
1     1     1
2     2     4
3     6     4
4     10    1
5     11    20
6     31    1
7     32    30
8     62    35
9     97    35
10    132   35
11    167   4
12    171   16
13    187   35
14    222   4
15    226   16
16    242   20
17    262   20
18    282   20
19    302   30
20    332   4
21    336   4
22    340   1
23    341   8
24    349   8
25    357   8
26    365   2
27    367   2
28    369   4
29    373   4
30    377   1
31    378   2
32    380   8
33    388   4
34    392   4
35    396   4
36    400   4
```

```
37    404   1
38    405   4
39    409   4
40    413   4
41    417   4
42    421   4
43    425   4
44    429   20
45    449   30
46    479   1
47    480   1
48    481   79
49    560   79
50    639   79
51    718   79
52    797   8
53    805   1
54    806   1
55    807   20
56    827   4
57    831   4

shell62; pack_isam station.ISM
Compressing station.ISM: (1192 records)
− Calculating statistics

normal:     20  empty−space:      16  empty−zero:        12  empty−fill:  11
pre−space:   0  end−space:        12  table−lookups:      5  zero:         7
Original trees:  57  After join: 17
− Compressing file
87.14%

shell62; ls −l station.*
−rw−rw−r−−   1 monty    my         127874 Apr 17 19:00 station.ISD
−rw−rw−r−−   1 monty    my          55296 Apr 17 19:04 station.ISM
−rw−rw−r−−   1 monty    my           5767 Apr 17 19:00 station.frm

shell62; isamchk −dvv station

ISAM file:     station
Isam−version:  2
Creation time: 1996−03−13 10:08:58
Recover time:  1997−04−17 19:04:26
Data records:              1192  Deleted blocks:              0
Datafile: Parts:           1192  Deleted data:                0
Datafilepointer (bytes):      3  Keyfile pointer (bytes):     1
Max datafile length:   16777215  Max keyfile length:     131071
Recordlength:               834
Record format: Compressed

table description:
Key Start Len Index   Type                      Root  Blocksize    Rec/key
1   2     4   unique  unsigned long            10240       1024          1
2   32    30  multip. text                     54272       1024          1

Field Start Length Type                       Huff tree  Bits
1     1     1      constant                           1     0
2     2     4      zerofill(1)                        2     9
3     6     4      no zeros, zerofill(1)              2     9
4     10    1                                         3     9
5     11    20     table−lookup                       4     0
6     31    1                                         3     9
```

```
7     32    30     no endspace, not_always              5     9
8     62    35     no endspace, not_always, no empty    6     9
9     97    35     no empty                             7     9
10    132   35     no endspace, not_always, no empty    6     9
11    167   4      zerofill(1)                          2     9
12    171   16     no endspace, not_always, no empty    5     9
13    187   35     no endspace, not_always, no empty    6     9
14    222   4      zerofill(1)                          2     9
15    226   16     no endspace, not_always, no empty    5     9
16    242   20     no endspace, not_always              8     9
17    262   20     no endspace, no empty                8     9
18    282   20     no endspace, no empty                5     9
19    302   30     no endspace, no empty                6     9
20    332   4      always zero                          2     9
21    336   4      always zero                          2     9
22    340   1                                           3     9
23    341   8      table−lookup                         9     0
24    349   8      table−lookup                        10     0
25    357   8      always zero                          2     9
26    365   2                                           2     9
27    367   2      no zeros, zerofill(1)                2     9
28    369   4      no zeros, zerofill(1)                2     9
29    373   4      table−lookup                        11     0
30    377   1                                           3     9
31    378   2      no zeros, zerofill(1)                2     9
32    380   8      no zeros                             2     9
33    388   4      always zero                          2     9
34    392   4      table−lookup                        12     0
35    396   4      no zeros, zerofill(1)               13     9
36    400   4      no zeros, zerofill(1)                2     9
37    404   1                                           2     9
38    405   4      no zeros                             2     9
39    409   4      always zero                          2     9
40    413   4      no zeros                             2     9
41    417   4      always zero                          2     9
42    421   4      no zeros                             2     9
43    425   4      always zero                          2     9
44    429   20     no empty                             3     9
45    449   30     no empty                             3     9
46    479   1                                          14     4
47    480   1                                          14     4
48    481   79     no endspace, no empty               15     9
49    560   79     no empty                             2     9
50    639   79     no empty                             2     9
51    718   79     no endspace                         16     9
52    797   8      no empty                             2     9
53    805   1                                          17     1
54    806   1                                           3     9
55    807   20     no empty                             3     9
56    827   4      no zeros, zerofill(2)                2     9
57    831   4      no zeros, zerofill(1)                2     9
```

The information printed by `pack_isam` is described below:

`normal`
: The number of columns for which no extra packing is used.

`empty−space`
: The number of columns containing values that are only spaces; these will occupy 1 bit.

`empty−zero`
: The number of columns containing values that are only binary 0's; these will occupy 1 bit.

`empty−fill`
: The number of integer columns that don't occupy the full byte range of their type; these are changed to a smaller type (for example, an `INTEGER` column may be changed to `MEDIUMINT`).

pre−space
: The number of decimal columns that are stored with leading space. In this case, each value will contain a count for the number of leading spaces.

end−space
: The number of columns that have a lot of trailing space. In this case, each value will contain a count for the number of trailing spaces.

table−lookup
: The column had only a small number of different values, and that were converted to an `ENUM` before Huffman compression.

zero
: The number of columns for which all values are zero.

Original trees
: The initial number of Huffman trees.

After join
: The number of distinct Huffman trees left after joining trees to save some header space.

After a table has been compressed, `isamchk −dvv` prints additional information about each field:

Type
: The field type may contain the following descriptors:

  constant
  : All rows have the same value.

  no endspace
  : Don't store endspace.

  no endspace, not_always
  : Don't store endspace and don't do end space compression for all values.

  no endspace, no empty
  : Don't store endspace. Don't store empty values.

  table−lookup
  : The column was converted to an `ENUM`.

  zerofill(n)
  : The most significant `n` bytes in the value are always 0 and are not stored.

  no zeros
  : Don't store zeros.

  always zero
  : 0 values are stored in 1 bit.

Huff tree
: The Huffman tree associated with the field

Bits
: The number of bits used in the Huffman tree.

13 Maintenance d'un serveur MySQL

13.1 Utiliser `isamchk` pour la maintenance et la réparation

Pour vérifier/réparer les tables ISAM (`.ISM` et `.ISD`), vous devez utiliser l'utilitaire : `isamchk`. Pour vérifier/réparer les tables MyISAM (`.MYI` et `.MYD`) vous devez utiliser utiliser l'utilitaire `myisamchk`. 10.18 Types de tables MySQL.

Par la suite, nous traiterons le cas de `isamchk` mais les instructions pourront aussi s'appliquer ˆ `myisamchk`.

Vous pouvez utiliser l'utilitaire `isamchk` pour receuillir des informations sur votre base de données et ses tables, pour vérifier et réparer les tables ou encore les optimiser. La section suivante décrit les commandes d'invocations de `isamchk` (y compris les options), les méthodes de maintenance régulière, et comment utiliser `isamchk` pour effectuer diverses tàches.

If you run `mysqld` with `--skip-locking` (which is the default on some systems, like Linux), you can't reliably use `isamchk` to check a table when `mysqld` is using the same table. If you can be sure that no one is accessing the tables through `mysqld` while you run `isamchk`, you only have to do `mysqladmin flush-tables` before you start checking the tables. If you can't guarantee the above, then you must take down `mysqld` while you check the tables. If you run `isamchk` while `mysqld` is updating the tables, you may get a warning that a table is corrupt even if it isn't.

If you are not using `--skip-locking`, you can use `isamchk` to check tables at any time. While you do this, all clients that try to update the table will wait until `isamchk` is ready before continuing.

If you use `isamchk` to repair or optimize tables, you ***MUST*** always ensure that the `mysqld` server is not using the table (this also applies if you are using `--skip-locking`). If you don't take down `mysqld` you should at least do a `mysqladmin flush-tables` before you run `isamchk`.

You can in most cases also use the command `OPTIMIZE TABLES` to optimize and repair tables, but this is not as fast or reliable (in case of real fatal errors) as `isamchk`. On the other hand, `OPTIMIZE TABLE` is easier to use and you don't have to worry about flushing tables. `OPTIMIZE TABLE`.

13.1.1 Syntaxe `isamchk`

`isamchk` is invoked like this:

```
shell62; isamchk [options] nom_table
```

The `options` specify what you want `isamchk` to do. They are described below. (You can also get a list of options by invoking `isamchk --help`.) With no options, `isamchk` simply checks your table. To get more information or to tell `isamchk` to take corrective action, specify options as described below and in the following sections.

`nom_table` is the database table you want to check. If you run `isamchk` somewhere other than in the database directory, you must specify the path to the file, since `isamchk` has no idea where your database is located. Actually, `isamchk` doesn't care whether or not the files you are working on are located in a database directory; you can copy the files that correspond to a database table into another location and

perform recovery operations on them there.

You can name several tables on the `isamchk` command line if you wish. You can also specify a name as an index file name (with the `.ISM'` suffix), which allows you to specify all tables in a directory by using the pattern `*.ISM'`. For example, if you are in a database directory, you can check all the tables in the directory like this:

```
shell62; isamchk *.ISM
```

If you are not in the database directory, you can check all the tables there by specifying the path to the directory:

```
shell62; isamchk /path/to/database_dir/*.ISM
```

You can even check all tables in all databases by specifying a wildcard with the path to the ***MySQL*** data directory:

```
shell62; isamchk /path/to/datadir/*/*.ISM
```

`isamchk` supports the following options:

−a, −−analyze
: Analyze the distribution of keys. This improves join performance by enabling the join optimizer to better choose in which order it should join the tables and which keys it should how use.

−#, −−debug=debug_options
: Output debug log. The `debug_options` string often is `'d:t:o,filename'`.

−d, −−description
: Prints some information about the table.

−e, −−extend−check
: Check the table VERY thoroughly. This is necessary only in extreme cases. Normally, `isamchk` should find all errors even without this option.

−f, −−force
: Overwrite old temporary files. If you use `−f` when checking tables (running `isamchk` without `−r`), `isamchk` will automatically restart with `−r` on any table for which an error occurs during checking.

−−help
: Display a help message and exit.

−i, −−information
: Print informational statistics about the table that is checked.

−k #, −−keys−used=#
: Used with `−r`. Tell the NISAM table handler to update only the first # indexes. Higher−numbered indexes are deactivated. This can be used to get faster inserts! Deactivated indexes can be reactivated by using `isamchk −r`.

−l, −−no−symlinks
: Do not follow symbolic links when repairing. Normally `isamchk` repairs the table a symlink points at.

−q, −−quick
: Used with `−r` to get a faster repair. Normally, the original data file isn't touched; you can specify a second `−q` to force the original data file to be used.

−r, −−recover
: Recovery mode. Can fix almost anything except unique keys that aren't unique.

−o, −−safe−recover
: Recovery mode. Uses an old recovery méthode; this is slower than `−r`, but can handle a couple of cases that `−r` cannot handle.

−O var=option, −−set−variable var=option
: Set the value of a variable. The possible variables are listed below.

−s, −−silent
: Silent mode. Write output only when errors occur. You can use `−s` twice (`−ss`) to make `isamchk` very silent.

−S, −−sort−index
: Sort the index tree blocks in high−low order. This will optimize seeks and will make table scanning by key faster.

−R index_num, −−sort−records=index_num
: Sorts records according to an index. This makes your data much more localized and may speed up ranged `SELECT` and `ORDER BY` operations on this index. (It may be VERY slow to do a sort the first time!) To find out a table's index numbers, use `SHOW INDEX`, which shows a table's indexes in the same order that `isamchk` sees them. Indexes are numbered beginning with 1.

−u, −−unpack
: Unpack a table that was packed with `pack_isam`.

−v, −−verbose

Verbose mode. Print more information. This can be used with `−d` and `−e`. Use `−v` multiple times (`−vv`, `−vvv`) for more verbosity!

`−V, −−version`
Print the `isamchk` version and exit.

`−w, −−wait`
Wait if the table is locked.

Possible variables for the `−−set−variable` (`−O`) option are:

```
key_buffer_size       current value: 16776192
read_buffer_size      current value: 262136
write_buffer_size     current value: 262136
sort_buffer_size      current value: 2097144
sort_key_blocks       current value: 16
decode_bits           current value: 9
```

13.1.2 Utilisation de la mémoire par `isamchk`

Memory allocation is important when you run `isamchk`. `isamchk` uses no more memory than you specify with the `−O` options. If you are going to use `isamchk` on very large files, you should first decide how much memory you want it to use. The default is to use only about 3M to fix things. By using larger values, you can get `isamchk` to operate faster. For example, if you have more than 32M RAM, you could use options such as these (in addition to any other options you might specify):

```
shell62; isamchk −O sort=16M −O key=16M −O read=1M −O write=1M ...
```

Using `−O sort=16M` should probably be enough for most cases.

Be aware that `isamchk` uses temporary files in `TMPDIR`. If `TMPDIR` points to a memory file system, you may easily get out of memory errors. If this happens, set `TMPDIR` to point at some directory with more space and restart `isamchk`

13.2 Mettre en place un régime de maintenance

It is a good idea to perform table checks on a regular basis rather than waiting for problems to occur. For maintenance purposes, you can use `isamchk −s` to check tables. The `−s` option causes `isamchk` to run in silent mode, printing messages only when errors occur.

It's a good idea to check tables when the server starts up. For example, whenever the machine has done a reboot in the middle of an update, you usually need to check all the tables that could have been affected. (This is an ``expected crashed table''.) You could add a test to `safe_mysqld` that runs `isamchk` to check all tables that have been modified during the last 24 hours if there is an old `` `.pid' `` (process ID) file left after a reboot. (The `` `.pid' `` file is created by `mysqld` when it starts up and removed when it terminates normally. The presence of a `` `.pid' `` file at system startup time indicates that `mysqld` terminated abnormally.)

An even better test would be to check any table whose last−modified time is more recent than that of the `` `.pid' `` file.

You should also check your tables regularly during normal system operation. At TcX, we run a `cron` job to check all our important tables once a week, using a line like this in a `` `crontab' `` file:

```
35 0 * * 0 /path/to/isamchk −s /path/to/datadir/*/*.ISM
```

This prints out information about crashed tables so we can examine and repair them when needed.

As we haven't had any unexpectedly crashed tables (tables that become corrupted for reasons other than hardware trouble) for a couple of years now (this is really true), once a week is more than enough for us.

We recommend that to start with, you execute `isamchk −s` each night on all tables that have been updated during the last 24 hours, until you come to trust ***MySQL*** as much as we do.

13.3 Informations sur une table

To get a description of a table or statistics about it, use the commands shown below. We explain some of the information in more detail later.

`isamchk −d nom_table`
: Runs `isamchk` in ``describe mode'' to produce a description of your table. If you start the ***MySQL*** server using the `−−skip−locking` option, `isamchk` may report an error for a table that is updated while it runs. However, since `isamchk` doesn't change the table in describe mode, there isn't any risk of destroying data.

`isamchk −d −v nom_table`
: To produce more information about what `isamchk` is doing, add `−v` to tell it to run in verbose mode.

`isamchk −eis nom_table`
: Shows only the most important information from a table. It is slow since it must read the whole table.

`isamchk −eiv nom_table`
: This is like `−eis`, but tells you what is being done.

Example of `isamchk −d` output:

```
ISAM file:     company.ISM
Data records:           1403698  Deleted blocks:         0
Recordlength:               226
Record format: Fixed length

table description:
Key Start Len Index   Type
1   2     8   unique  double
2   15    10  multip. text packed stripped
3   219   8   multip. double
4   63    10  multip. text packed stripped
5   167   2   multip. unsigned short
6   177   4   multip. unsigned long
7   155   4   multip. text
8   138   4   multip. unsigned long
9   177   4   multip. unsigned long
    193   1           text
```

Example of `isamchk −d −v` output:

```
ISAM file:     company.ISM
Isam−version:  2
Creation time: 1996−08−28 11:44:22
Recover time:  1997−01−12 18:35:29
Data records:           1403698  Deleted blocks:              0
Datafile: Parts:        1403698  Deleted data:                0
Datafilepointer (bytes):      3  Keyfile pointer (bytes):     3
Max datafile length: 3791650815  Max keyfile length: 4294967294
Recordlength:               226
Record format: Fixed length

table description:
```

```
Key Start Len Index   Type                     Root Blocksize Rec/key
1   2     8   unique  double                15845376      1024       1
2   15    10  multip. text packed stripped  25062400      1024       2
3   219   8   multip. double                40907776      1024      73
4   63    10  multip. text packed stripped  48097280      1024       5
5   167   2   multip. unsigned short        55200768      1024    4840
6   177   4   multip. unsigned long         65145856      1024    1346
7   155   4   multip. text                  75090944      1024    4995
8   138   4   multip. unsigned long         85036032      1024      87
9   177   4   multip. unsigned long         96481280      1024     178
    193   1           text
```

Example of `isamchk −eis` output:

```
Checking ISAM file: company.ISM
Key:  1:  Keyblocks used:  97%  Packed:    0%  Max levels:  4
Key:  2:  Keyblocks used:  98%  Packed:   50%  Max levels:  4
Key:  3:  Keyblocks used:  97%  Packed:    0%  Max levels:  4
Key:  4:  Keyblocks used:  99%  Packed:   60%  Max levels:  3
Key:  5:  Keyblocks used:  99%  Packed:    0%  Max levels:  3
Key:  6:  Keyblocks used:  99%  Packed:    0%  Max levels:  3
Key:  7:  Keyblocks used:  99%  Packed:    0%  Max levels:  3
Key:  8:  Keyblocks used:  99%  Packed:    0%  Max levels:  3
Key:  9:  Keyblocks used:  98%  Packed:    0%  Max levels:  4
Total:    Keyblocks used:  98%  Packed:   17%

Records:          1403698    M.recordlength:     226   Packed:             0%
Recordspace used:     100%   Empty space:          0%  Blocks/Record:   1.00
Recordblocks:     1403698    Deleteblocks:         0
Recorddata:     317235748    Deleted data:         0
Lost space:             0    Linkdata:             0

User time 1626.51, System time 232.36
Maximum resident set size 0, Integral resident set size 0
Non physical pagefaults 0, Physical pagefaults 627, Swaps 0
Blocks in 0 out 0, Messages in 0 out 0, Signals 0
Voluntary context switches 639, Involuntary context switches 28966
```

Example of `isamchk −eiv` output:

```
Checking ISAM file: company.ISM
Data records: 1403698   Deleted blocks:       0
− check file−size
− check delete−chain
index  1:
index  2:
index  3:
index  4:
index  5:
index  6:
index  7:
index  8:
index  9:
No recordlinks
− check index reference
− check data record references index: 1
Key:  1:  Keyblocks used:  97%  Packed:    0%  Max levels:  4
− check data record references index: 2
Key:  2:  Keyblocks used:  98%  Packed:   50%  Max levels:  4
− check data record references index: 3
Key:  3:  Keyblocks used:  97%  Packed:    0%  Max levels:  4
```

```
− check data record references index: 4
Key:  4:  Keyblocks used:  99%  Packed:   60%  Max levels:  3
− check data record references index: 5
Key:  5:  Keyblocks used:  99%  Packed:    0%  Max levels:  3
− check data record references index: 6
Key:  6:  Keyblocks used:  99%  Packed:    0%  Max levels:  3
− check data record references index: 7
Key:  7:  Keyblocks used:  99%  Packed:    0%  Max levels:  3
− check data record references index: 8
Key:  8:  Keyblocks used:  99%  Packed:    0%  Max levels:  3
− check data record references index: 9
Key:  9:  Keyblocks used:  98%  Packed:    0%  Max levels:  4
Total:    Keyblocks used:   9%  Packed:   17%

− check records and index references
[LOTS OF ROW NUMBERS DELETED]

Records:          1403698    M.recordlength:     226   Packed:             0%
Recordspace used:     100%   Empty space:          0%  Blocks/Record:   1.00
Recordblocks:     1403698    Deleteblocks:         0
Recorddata:     317235748    Deleted data:         0
Lost space:             0    Linkdata:             0

User time 1639.63, System time 251.61
Maximum resident set size 0, Integral resident set size 0
Non physical pagefaults 0, Physical pagefaults 10580, Swaps 0
Blocks in 4 out 0, Messages in 0 out 0, Signals 0
Voluntary context switches 10604, Involuntary context switches 122798
```

Here are the sizes of the data and index files for the table used in the preceding examples:

```
−rw−rw−r−−   1 monty    tcx     317235748 Jan 12 17:30 company.ISD
−rw−rw−r−−   1 davida   tcx      96482304 Jan 12 18:35 company.ISM
```

Explanations for the types of information `isamchk` produces are given below. The ``keyfile'' is the index file. ``Record'' and ``row'' are synonymous.

`ISAM file`
: Name of the ISAM (index) file.

`Isam−version`
: Version of ISAM format. Currently always 2.

`Creation time`
: When the data file was created.

`Recover time`
: When the index/data file was last reconstructed.

`Data records`
: How many records are in the table.

`Deleted blocks`
: How many deleted blocks still have reserved space. You can optimize your table to minimize this space. 13.4.3 Optimisation de tables.

`Datafile: Parts`
: For dynamic record format, this indicates how many data blocks there are. For an optimized table without fragmented records, this is the same as `Data records`.

`Deleted data`
: How many bytes of non−reclaimed deleted data there are. You can optimize your table to minimize this space. 13.4.3 Optimisation de tables.

`Datafile pointer`
: The size of the data file pointer, in bytes. It is usually 2, 3, 4 or 5 bytes. Most tables manage with 2 bytes, but this cannot be controlled from ***MySQL*** yet. For fixed tables, this is a record address. For dynamic tables, this is a byte address.

`Keyfile pointer`
: The size of the index file pointer, in bytes. It is usually 1, 2 or 3 bytes. Most tables manage with 2 bytes, but this is calculated automatically by ***MySQL***. It is always a block address.

`Max datafile length`
: How long the table's data file (`.ISD` file) can become, in bytes.

`Max keyfile length`

How long the table's key file (`.ISM` file) can become, in bytes.

`Recordlength`
How much space each record takes, in bytes.

`Record format`
The format used to store table rows. The examples shown above use `Fixed length`. Other possible values are `Compressed` and `Packed`.

`table description`
A list of all keys in the table. For each key, some low−level information is presented:

`Key`
This key's number.

`Start`
Where in the record this index part starts.

`Len`
How long this index part is. For packed numbers, this should always be the full length of the column. For strings, it may be shorter than the full length of the indexed column, because you can index a prefix of a string column.

`Index`
`unique` or `multip.` (multiple). Indicates whether or not one value can exist multiple times in this index.

`Type`
What data−type this index part has. This is an NISAM data−type with the options `packed`, `stripped` or `empty`.

`Root`
Address of the root index block.

`Blocksize`
The size of each index block. By default this is 1024, but the value may be changed at compile time.

`Rec/key`
This is a statistical value used by the optimizer. It tells how many records there are per value for this key. A unique key always has a value of 1. This may be updated after a table is loaded (or greatly changed) with `isamchk −a`. If this is not updated at all, a default value of 30 is given.

In the first example above, the 9th key is a multi−part key with two parts.

`Keyblocks used`
What percentage of the keyblocks are used. Since the table used in the examples had just been reorganized with `isamchk`, the values are very high (very near the theoretical maximum).

`Packed`
MySQL tries to pack keys with a common suffix. This can only be used for `CHAR`/`VARCHAR`/`DECIMAL` keys. For long strings like names, this can significantly reduce the space used. In the third example above, the 4th key is 10 characters long and a 60% reduction in space is achieved.

`Max levels`
How deep the B−tree for this key is. Large tables with long keys get high values.

`Records`
How many rows are in the table.

`M.recordlength`
The average record length. For tables with fixed−length records, this is the exact record length.

`Packed`
MySQL strips spaces from the end of strings. The `Packed` value indicates the percentage savings achieved by doing this.

`Recordspace used`
What percentage of the data file is used.

`Empty space`
What percentage of the data file is unused.

`Blocks/Record`
Average number of blocks per record (i.e., how many links a fragmented record is composed of). This is always 1 for fixed−format tables. This value should stay as close to 1.0 as possible. If it gets too big, you can reorganize the table with `isamchk`. 13.4.3 Optimisation de tables.

`Recordblocks`
How many blocks (links) are used. For fixed format, this is the same as the number of records.

`Deleteblocks`
How many blocks (links) are deleted.

`Recorddata`
How many bytes in the data file are used.

`Deleted data`
How many bytes in the data file are deleted (unused).

`Lost space`
If a record is updated to a shorter length, some space is lost. This is the sum of all such losses, in bytes.

`Linkdata`
When the dynamic table format is used, record fragments are linked with pointers (4 to 7 bytes each). `Linkdata` is the sum of the amount of storage used by all such pointers.

If a table has been compressed with `pack_isam`, `isamchk −d` prints additional information about each table column. See `pack isam`, for an example of this information and a description of what it means.

13.4 Utiliser isamchk pour réparer une table

The file format that ***MySQL*** uses to store data has been extensively tested, but there are always external circumstances that may cause database tables to become corrupted:

- The `mysqld` process being killed in the middle of a write
- Unexpected shutdown of the computer (for example, if the computer is turned off)
- A hardware error

This chapter describes how to check for and deal with data corruption in ***MySQL*** databases. If your tables get corrupted a lot you should try to find the reason for this! G.1 Debugguer un serveur MySQL.

When performing crash recovery, it is important to understand that each table `nom_table` in a database corresponds to three files in the database directory:

File	*Purpose*
`` `nom_table.frm' ``	Table definition (form) file
`` `nom_table.ISD' ``	Data file
`` `nom_table.ISM' ``	Index file

Each of these three file types is subject to corruption in various ways, but problems occur most often in data files and index files.

`isamchk` works by creating a copy of the `` `.ISD' `` (data) file row by row. It ends the repair stage by removing the old `` `.ISD' `` file and renaming the new file to the original file name. If you use `--quick`, `isamchk` does not create a temporary `` `.ISD' `` file, but instead assumes that the `` `.ISD' `` file is correct and only generates a new index file without touching the `` `.ISD' `` file. This is safe, because `isamchk` automatically detects if the `` `.ISD' `` file is corrupt and aborts the repair in this case. You can also give two `--quick` options to `isamchk`. In this case, `isamchk` does not abort on some errors (like duplicate key) but instead tries to resolve them by modifying the `` `.ISD' `` file. Normally the use of two `--quick` options is useful only if you have too little free disk space to perform a normal repair. In this case you should at least make a backup before running `isamchk`.

13.4.1 Comment vérifier une table

To check a table, use the following commands:

`isamchk nom_table`
: This finds 99.99% of all errors. What it can't find is corruption that involves ***ONLY*** the data file (which is very unusual). If you want to check a table, you should normally run `isamchk` without options or with either the `-s` or `--silent` option.

`isamchk -e nom_table`
: This does a complete and thorough check of all data (`-e` means ``extended check''). It does a check−read of every key for each row to verify that they indeed point to the correct row. This may take a LONG time on a big table with many keys. `isamchk` will normally stop after the first error it finds. If you want to obtain more information, you can add the `--verbose` (`-v`) option. This causes `isamchk` to keep going, up through a maximum of 20 errors. In normal usage, a simple `isamchk` (with no arguments other than the table name) is sufficient.

`isamchk -e -i nom_table`
: Like the previous command, but the `-i` option tells `isamchk` to print some informational statistics, too.

13.4.2 Comment réparer une table

The symptoms of a corrupted table are usually that requêtesabort unexpectedly and that you observe errors such as these:

- `nom_table.frm' is locked against change
- Can't find file `nom_table.ISM' (Errcode: ###)
- Got error ### from table handler (Error 135 is an exception in this case)
- Unexpected end of file
- Record file is crashed

In these cases, you must repair your tables. `isamchk` can usually detect and fix most things that go wrong.

The repair process involves up to four stages, described below. Before you begin, you should `cd` to the database directory and check the permissions of the table files. Make sure they are readable by the Unix user that `mysqld` runs as (and to you, since you need to access the files you are checking). If it turns out you need to modify files, they must also be writable by you.

Stage 1: Checking your tables

Run `isamchk *.ISM` or (`isamchk −e *.ISM` if you have more time). Use the `−s` (silent) option to suppress unnecessary information.

You have to repair only those tables for which `isamchk` announces an error. For such tables, proceed to Stage 2.

If you get weird errors when checking (such as `out of memory` errors), or if `isamchk` crashes, go to Stage 3.

Stage 2: Easy safe repair

First, try `isamchk −r −q nom_table` (`−r −q` means ``quick recovery mode''). This will attempt to repair the index file without touching the data file. If the data file contains everything that it should and the delete links point at the correct locations within the data file, this should work and the table is fixed. Start repairing the next table. Otherwise, use the following procedure:

1. Make a backup of the data file before continuing.
2. Use `isamchk −r nom_table` (`−r` means ``recovery mode''). This will remove incorrect records and deleted records from the data file and reconstruct the index file.
3. If the preceding step fails, use `isamchk −−safe−recover nom_table`. Safe recovery mode uses an old recovery méthode that handles a few cases that regular recovery mode doesn't (but is slower).

If you get weird errors when repairing (such as `out of memory` errors), or if `isamchk` crashes, go to Stage 3.

Stage 3: Difficult repair

You should only reach this stage if the first 16K block in the index file is destroyed or contains incorrect information, or if the index file is missing. In this case, it's necessary to create a new index file. Do so as follows:

1. Move the data file to some safe place.
2. Use the table description file to create new (empty) data and index files:
   ```
   shell62; mysql nom_base_de_donnees
   ```

```
mysql62; DELETE FROM nom_table;
mysql62; quit
```

3. Copy the old data file back onto the newly created data file. (Don't just move the old file back onto the new file; you want to retain a copy in case something goes wrong.)

Go back to Stage 2. `isamchk −r −q` should work now. (This shouldn't be an endless loop).

Stage 4: Very difficult repair

You should reach this stage only if the description file has also crashed. That should never happen, because the description file isn't changed after the table is created.

1. Restore the description file from a backup and go back to Stage 3. You can also restore the index file and go back to Stage 2. In the latter case, you should start with `isamchk −r`.
2. If you don't have a backup but know exactly how the table was created, create a copy of the table in another database. Remove the new data file, then move the description and index files from the other database to your crashed database. This gives you new description and index files, but leaves the data file alone. Go back to Stage 2 and attempt to reconstruct the index file.

13.4.3 Optimisation de tables

To coalesce fragmented records and eliminate wasted space resulting from deleting or updating records, run `isamchk` in recovery mode:

```
shell62; isamchk −r nom_table
```

You can optimize a table in the same way using the SQL `OPTIMIZE TABLE` statement. `OPTIMIZE TABLE` is easier, but `isamchk` is faster.

`isamchk` also has a number of other options you can use to improve the performance of a table:

```
−S, −−sort−index
−R index_num, −−sort−records=index_num
−a, −−analyze
```

For a full description of the option see 13.1.1 Syntaxe `isamchk`.

13.5 Maintenance du fichier d'historique

When using ***MySQL*** with log files, you will from time to time want to remove/backup old log files and tell ***MySQL*** to start logging on new files. 9.2 Historique de modification.

One a Linux (`Redhat`) installation, you can use the `mysql−log−rotate` script for this. If you installed ***MySQL*** from an RPM distribution, the script should have been installed automatically.

On other systems you must install a short script yourself that you start from `cron` to handle log files.

You can force ***MySQL*** to start using new log files by using `mysqladmin flush−logs` or by using the SQL command `FLUSH LOGS`. If you are using ***MySQL*** 3.21 you must use `mysqladmin refresh`.

The above command does the following:

- If standard logging (`−−log`) is used, closes and reopens the log file. (`` `mysql.log' `` as default).
- If update logging (`−−log−update`) is used, closes the update log and opens a new log file with a higher sequence number.

If you are using only an update log, you only have to flush the logs and then move away the old update log files to a backup. If you are using the normal logging, you can do something like:

```
shell62; cd mysql−data−directory
shell62; mv mysql.log mysql.old
shell62; mysqladmin flush−tables
```

and then take a backup and remove `mysql.old'.

14 Ajouter des fonctions à MySQL

There are two ways to add new functions to ***MySQL***:

- You can add the function through the user−definable function (UDF) interface. User−definable functions are added and removed dynamically using the `CREATE FUNCTION` and `DROP FUNCTION` statements. `CREATE FUNCTION`.
- You can add the function as a native (built in) ***MySQL*** function. Native functions are compiled into the `mysqld` server and become available on a permanent basis.

Each méthode has advantages and disadvantages:

- If you write a user−definable function, you must install the object file in addition to the server itself. If you compile your function into the server, you don't need to do that.
- You can add UDFs to a binary ***MySQL*** distribution. Native functions require you to modify a source distribution.
- If you upgrade your ***MySQL*** distribution, you can continue to use your previously−installed UDFs. For native functions, you must repeat your modifications each time you upgrade.

Whichever méthode you use to add new functions, they may be used just like native functions such as `ABS()` or `SOUNDEX()`.

14.1 Ajouter une nouvelle fonction utilisateur

For the UDF mechanism to work, functions must be written in C or C++ and your operating system must support dynamic loading. The ***MySQL*** source distribution includes a file `` `sql/udf_example.cc' `` that defines 5 new functions. Consult this file to see how UDF calling conventions work.

For each function that you want to use in SQL statements, you should define corresponding C (or C++) functions. In the discussion below, the name ``xxx'' is used for an example function name. To distinquish between SQL and C/C++ usage, `XXX()` (uppercase) indicates a SQL function call, and `xxx()` (lowercase) indicates a C/C++ function call.

The C/C++ functions that you write to implement the inferface for `XXX()` are:

`xxx()` (required)
The main function. This is where the function result is computed. The correspondence between the SQL type and return type of your C/C++ function is shown below:

SQL type	***C/C++ type***
`STRING`	`char *`
`INTEGER`	`long long`
`REAL`	`double`

`xxx_init()` (optional)
The initialization function for `xxx()`. It can be used to:
◊ Check the number of arguments to `XXX()`
◊ Check that the arguments are of a required type, or, alternatively, tell ***MySQL*** to coerce arguments to the types you want when the main function is called
◊ Allocate any memory required by the main function
◊ Specify the maximum length of the result
◊ Specify (for `REAL` functions) the maximum number of decimals
◊ Specify whether or not the result can be `NULL`

`xxx_deinit()` (optional)
The deinitialization function for `xxx()`. It should deallocate any memory allocated by the initialization function.

When a SQL statement invokes `XXX()`, ***MySQL*** calls the initialization function `xxx_init()` to let it perform any required setup, such as argument checking or memory allocation. If `xxx_init()` returns an

error, the SQL statement is aborted with an error message and the main and deinitialization functions are not called. Otherwise, the main function `xxx()` is called once for each row. After all rows have been processed, the deinitialization function `xxx_deinit()` is called so it can perform any required cleanup.

All functions must be thread−safe (not just the main function, but the initialization and deinitialization functions as well). This means that you are not allowed to allocate any global or static variables that change! If you need memory, you should allocate it in `xxx_init()` and free it in `xxx_deinit()`.

14.1.1 Séquences UDF

The main function should be declared as shown below. Note that the return type and parameters differ, depending on whether you will declare the SQL function `XXX()` to return `STRING`, `INTEGER` or `REAL` in the `CREATE FUNCTION` statement:

For `STRING` functions:

```
char *xxx(UDF_INIT *initid, UDF_ARGS *args,
              char *result, unsigned long *length,
              char *is_null, char *error);
```

For `INTEGER` functions:

```
long long xxx(UDF_INIT *initid, UDF_ARGS *args,
              char *is_null, char *error);
```

For `REAL` functions:

```
double xxx(UDF_INIT *initid, UDF_ARGS *args,
              char *is_null, char *error);
```

The initialization and deinitialization functions are declared like this:

```
my_bool xxx_init(UDF_INIT *initid, UDF_ARGS *args, char *message);

void xxx_deinit(UDF_INIT *initid);
```

The `initid` parameter is passed to all three functions. It points to a `UDF_INIT` structure that is used to communicate information between functions. The `UDF_INIT` structure members are listed below. The initialization function should fill in any members that it wishes to change. (To use the default for a member, leave it unchanged.)

`my_bool maybe_null`
: `xxx_init()` should set `maybe_null` to `1` if `xxx()` can return `NULL`. The default value is `1` if any of the arguments are declared `maybe_null`.

`unsigned int decimals`
: Number of decimals. The default value is the maximum number of decimals in the arguments passed to the main function. (For example, if the function is passed `1.34`, `1.345` and `1.3`, the default would be 3, since `1.345` has 3 decimals.

`unsigned int max_length`
: The maximum length of the string result. The default value differs depending on the result type of the function. For string functions, the default is the length of the longest argument. For integer functions, the default is 21 digits. For real functions, the default is 13 plus the number of decimals indicated by `initid->decimals`. (For numeric functions, the length includes any sign or decimal point characters.)

`char *ptr`
: A pointer that the function can use for its own purposes. For example, functions can use `initid->ptr` to communicate allocated memory between functions. In `xxx_init()`, allocate the memory and assign it to this pointer:
  `initid−62;ptr = allocated_memory;`

In `xxx()` and `xxx_deinit()`, refer to `initid->ptr` to use or deallocate the memory.

14.1.2 Traitement des arguments

The `args` parameter points to a `UDF_ARGS` structure which has the members listed below:

`unsigned int arg_count`
: The number of arguments. Check this value in the initialization function if you want your function to be called with a particular number of arguments. For example:

```
if (args−62;arg_count != 2)
{
    strcpy(message,"XXX() requires two arguments");
    return 1;
}
```

`enum Item_result *arg_type`
: The types for each argument. The possible type values are `STRING_RESULT`, `INT_RESULT` and `REAL_RESULT`. To make sure that arguments are of a given type and return an error if they are not, check the `arg_type` array in the initialization function. For example:

```
if (args−62;arg_type[0] != STRING_RESULT
      38;args−62;arg_type[1] != INT_RESULT)
{
    strcpy(message,"XXX() requires a string and an integer");
    return 1;
}
```

As an alternative to requiring your function's arguments to be of particular types, you can use the initialization function to set the `arg_type` elements to the types you want. This causes ***MySQL*** to coerce arguments to those types for each call to `xxx()`. For example, to specify coercion of the first two arguments to string and integer, do this in `xxx_init()`:

```
args−62;arg_type[0] = STRING_RESULT;
args−62;arg_type[1] = INT_RESULT;
```

`char **args`
: `args->args` communicates information to the initialization function about the general nature of the arguments your function was called with. For a constant argument `i`, `args->args[i]` points to the argument value. (See below for instructions on how to access the value properly.) For a non−constant argument, `args->args[i]` is `0`. A constant argument is an expression that uses only constants, such as `3` or `4*7−2` or `SIN(3.14)`. A non−constant argument is an expression that refers to values that may change from row to row, such as column names or functions that are called with non−constant arguments. For each invocation of the main function, `args->args` contains the actual arguments that are passed for the row currently being processed. Functions can refer to an argument `i` as follows:

  ◊ An argument of type `STRING_RESULT` is given as a string pointer plus a length, to allow handling of binary data or data of arbitrary length. The string contents are available as `args->args[i]` and the string length is `args->lengths[i]`. You should not assume that strings are null−terminated.

  ◊ For an argument of type `INT_RESULT`, you must cast `args->args[i]` to a `long long` value:

```
long long int_val;
int_val = *((long long*) args−62;args[i]);
```

  ◊ For an argument of type `REAL_RESULT`, you must cast `args->args[i]` to a `double` value:

```
double    real_val;
real_val = *((double*) args−62;args[i]);
```

`unsigned long *lengths`
: For the initialization function, the `lengths` array indicates the maximum string length for each argument. For each invocation of the main function, `lengths` contains the actual lengths of any string arguments that are passed for the row currently being processed. For arguments of types `INT_RESULT` or `REAL_RESULT`, `lengths` still contains the maximum length of the argument (as for the initialization function).

14.1.3 Valeurs retournées, et gestion des erreurs

The initialization function should return `0` if no error occurred and `1` otherwise. If an error occurs, `xxx_init()` should store a null−terminated error message in the `message` parameter. The message will be returned to the client. The message buffer is `MYSQL_ERRMSG_SIZE` characters long, but you should try to keep the message to less than 80 characters so that it fits the width of a standard terminal screen.

The return value of the main function `xxx()` is the function value, for `long long` and `double` functions. For string functions, the string is returned in the `result` and `length` arguments. `result` is a buffer at least 255 bytes long. Set these to the contents and length of the return value. For example:

```
memcpy(result, "result string", 13);
*length = 13;
```

The string function return value normally also points to the result.

To indicate a return value of `NULL` in the main function, set `is_null` to `1`:

```
*is_null = 1;
```

To indicate an error return in the main function, set the `error` parameter to `1`:

```
*error = 1;
```

If `xxx()` sets `*error` to `1` for any row, the function value is `NULL` for the current row and for any subsequent rows processed by the statement in which `XXX()` was invoked. (`xxx()` will not even be called for subsequent rows.) ***Note:*** In ***MySQL*** versions prior to 3.22.10, you should set both `*error` and `*is_null`:

```
*error = 1;
*is_null = 1;
```

14.1.4 Compilation et installation de fonction définies par l'utilisateur

Files implementing UDFs must be compiled and installed on the host where the server runs. This process is described below for the example UDF file `` `udf_example.cc' `` that is included in the ***MySQL*** source distribution. This file contains the following functions:

- `metaphon()` returns a metaphon string of the string argument. This is something like a soundex string, but it's more tuned for English.
- `myfunc_double()` returns the sum of the ASCII values of the characters in its arguments, divided by the sum of the length of its arguments.
- `myfunc_int()` returns the sum of the length of its arguments.
- `lookup()` returns the IP number for a hostname.
- `reverse_lookup()` returns the hostname for an IP number. The function may be called with a string `"xxx.xxx.xxx.xxx"` or four numbers.

A dynamically−loadable file should be compiled as a sharable object file, using a command something like this:

```
shell62; gcc −shared −o udf_example.so myfunc.cc
```

You can easily find out the correct compiler options for your system by running this command in the `` `sql' `` directory of your ***MySQL*** source tree:

```
shell62; make udf_example.o
```

You should run a compile command similar to the one that `make` displays, except that you should remove the `−c` option near the end of the line and add `−o udf_example.so` to the end of the line. (On some systems, you may need to leave the `−c` on the command.)

Once you compile a shared object containing UDFs, you must install it and tell ***MySQL*** about it. Compiling a

shared object from `udf_example.cc' produces a file named something like `udf_example.so' (the exact name may vary from platform to platform). Copy this file to some directory searched by ld, such as `/usr/lib'. On many systems, you can set the LD_LIBRARY or LD_LIBRARY_PATH environment variable to point at the directory where you have your UDF function files. The dopen manual page tells you which variable you should use on your system. You should set this in mysql.server or safe_mysqld and restart mysqld.

After the library is installed, notify mysqld about the new functions with these commands:

```
mysql62; CREATE FUNCTION metaphon RETURNS STRING SONAME "udf_example.so";
mysql62; CREATE FUNCTION myfunc_double RETURNS REAL SONAME "udf_example.so";
mysql62; CREATE FUNCTION myfunc_int RETURNS INTEGER SONAME "udf_example.so";
mysql62; CREATE FUNCTION lookup RETURNS STRING SONAME "udf_example.so";
mysql62; CREATE FUNCTION reverse_lookup RETURNS STRING SONAME "udf_example.so";
```

Functions can be deleted using DROP FUNCTION:

```
mysql62; DROP FUNCTION metaphon;
mysql62; DROP FUNCTION myfunc_double;
mysql62; DROP FUNCTION myfunc_int;
mysql62; DROP FUNCTION lookup;
mysql62; DROP FUNCTION reverse_lookup;
```

The CREATE FUNCTION and DROP FUNCTION statements update the system table func in the mysql database. The function's name, type and shared library name are saved in the table. You must have the ***insert*** and ***delete*** privileges for the mysql database to create and drop functions.

You should not use CREATE FUNCTION to add a function that has already been created. If you need to reinstall a function, you should remove it with DROP FUNCTION and then reinstall it with CREATE FUNCTION. You would need to do this, for example, if you recompile a new version of your function, so that mysqld gets the new version. Otherwise the server will continue to use the old version.

Active functions are reloaded each time the server starts, unless you start mysqld with the −−skip−grant−tables option. In this case, UDF initialization is skipped and UDFs are unavailable. (An active function is one that has been loaded with CREATE FUNCTION and not removed with DROP FUNCTION.)

14.2 Ajouter une nouvelle fonction native

The procedure for adding a new native function is described below. Note that you cannot add native functions to a binary distribution since the procedure involves modifying ***MySQL*** source code. You must compile ***MySQL*** yourself from a source distribution. Also note that if you migrate to another version of ***MySQL*** (e.g., when a new version is released), you will need to repeat the procedure with the new version.

To add a new native ***MySQL*** function, follow these steps:

1. Add one line to `lex.h' that defines the function name in the sql_functions[] array.
2. Add two lines to `sql_yacc.yy'. One indicates the preprocessor symbol that yacc should define (this should be added at the beginning of the file). Then define the function parameters and add an ``item" with these parameters to the simple_expr parsing rule. For an example, check all occurrences of SOUNDEX in `sql_yacc.yy' to see how this is done.
3. In `item_func.h', declare a class inheriting from Item_num_func or Item_str_func, depending on whether your function returns a number or a string.
4. In `item_func.cc', add one of the following declarations, depending on whether you are defining a numeric or string function:

```
double   Item_func_newname::val()
longlong Item_func_newname::val_int()
String  *Item_func_newname::Str(String *str)
```

5. You should probably also define the following function:
```
void Item_func_newname::fix_length_and_dec()
```

This function should at least calculate `max_length` based on the given arguments. `max_length` is the maximum number of characters the function may return. This function should also set `maybe_null = 0` if the main function can't return a `NULL` value. The function can check if any of the function arguments can return `NULL` by checking the arguments `maybe_null` variable.

All functions must be thread−safe.

For string functions, there are some additional considerations to be aware of:

- The `String *str` argument provides a string buffer that may be used to hold the result.
- The function should return the string that holds the result.
- All current string functions try to avoid allocating any memory unless absolutely necessary!

15 Ajouter de nouvelles procédures MySQL

Avec ***MySQL***, vous pouvez définir une procédure en C++ qui pourra accéder et modifier les données d'une requête, avant qu'elles ne soient envoyées au client. La modification peut être faite ligne par ligne, ou bien par groupe (avec `GROUP BY`).

Nous avons créé un exemple de procédure dans ***MySQL*** 3.23 pour montrer comment ca fonctionne :

15.1 Analyse de procédure

```
analyse([max elements,[max memory]])
```

Cette procédure est définie dans le fichier `` `sql/sql_analyse.cc' ``. Elle examine le résultat d'une requête et retourne cette analyse.

- `max elements` (par défaut 256) est le nombre maximum de valeurs distinctes que la procédure `analyse` va utiliser par colonne. `max elements` est utilisé par `analyse` pour s'assurer que le type optimal de la colonne devrait être `ENUM`.
- `max memory` (par défaut 8192) est la taille maximale de mémoire que `analyse` peut allouer par colonne, lorsqu'elle essaie de trouver des valeurs distinctes.

```
SELECT ... FROM ... WHERE ... PROCEDURE ANALYSE([max elements,[max memory]])
```

15.2 Ecrire une procédure.

Pour le moment, la seule documentation pour cette fonction est dans le source. :((et en anglais :( ).

Vous pouvez trouver d'autres informations concernant les procédures en examinant les fichiers suivants :

- `` `sql/sql_analyse.cc' ``
- `` `sql/procedure.h' ``
- `` `sql/procedure.cc' ``
- `` `sql/sql_select.cc' ``

16 MySQL et ODBC

MySQL fournit l'accès à ODBC gr ce au programme ***MyODBC***.

16.1 OS supportés par MyODBC

MyODBC est un pilote ODBC (2.50) 32−bit level 0 driver pour Windows95 et Windows NT. Nous esperons que quelqu'un voudra bien le porter sous Windows 3.x.

16.2 Comment rapporter des bugs avec MyODBC

MyODBC a été testé avec sucess sur Access, Admndemo.exe, C++−Builder, Centura Team Developer (feu Gupta SQL/Windows), ColdFusion (sous Solaris et NT service pack 5), Crystal Reports, DataJunction, Delphi, ERwin, Excel, iHTML, FileMaker Pro, FoxPro, Notes 4.5/4.6, SBSS, Perl DBD−ODBC, Paradox, Powerbuilder, Powerdesigner 32 bit, VC++ et Visual Basic.

Si vous connaissez d'autres applications qui fonctionnent avec ***MyODBC***, laissez nous un myodbc@lists.mysql.com !

Si vous rencontrez des difficultés, vous préférons avoir le fichier d'historique du gestionnaire ODBC ODBC (l'historique que vous obtenez en demandant l'historique de ODBCADMIN) et l'historique ***MyODBC*** . Cela nous éclairera sur vos problèmes.

Pour obtenir l'historique de ***MyODBC***, cochez l'option 'Trace MyODBC' dans l'écran de connexion/configuration de ***MyODBC***. L'historique sera écrit dans le fichier `C:\myodbc.log'. Notez que vous devez utiliser `MYSQL.DLL` et pas `MYSQL2.DLL` pour que cette option fonctionne.

16.3 Programmes testés avec MyODBC

La plupart des programmes fonctionnent avec ***MyODBC***, mais ceux de la liste suivante on été testé par nos soins, ou vérifié par quelqu'un qui l'utilise :

Program
: ***Comment***

Access
: Pour faire fonctionner Access :
  - ◊ Vous devez avoir une clé primaire dans votre table.
  - ◊ Vous devez avoir une colonne de type timestamp dans toutes les tables qui vous voudrez modifier.
  - ◊ N'utilisez que des champs double float. Access échoue de la la comparaison de simple floats.
  - ◊ Choisissez l'option 'Return matching rows' lors de la connexion à ***MySQL***.
  - ◊ Sous NT, Access retournera les colonnes de type `BLOB` comme des objets `OLE OBJECTS`. Si, à la place, vous voulez avoir des colonnes de types `MEMO`, il vous faut changer la colonne en `TEXT` avec `ALTER TABLE`.
  - ◊ Access ne gère pas toujours les `DATE` correctement. Si vous avez un souci avec, changez les colonnes en type `DATETIME`.
  - ◊ Dans certains cas, Access peut générer des requêtes SQL illégales, que ***MySQL*** ne peut pas comprendre. Vous pouvez corriger cela en selection l'option `"Query|SQLSpecific|Pass−Through"` dans le menu Access.

DataJunction
: Vous devez vous arranger pour retourner des colonnes de type `VARCHAR` plutot que `ENUM`, car il ne gère pas correctement ces dernières.

Excel
: Fonctionne. Conseils :
  - ◊ Si vous avez des problèmes avec les dates, essayez de les selectionner sous la forme de chaînes en utilisant la fonction `CONCAT()`. Par exemple :

```
select CONCAT(rise_time), CONCAT(set_time)
    from sunrise_sunset;
```

Les valeurs retournées sous la forme de chaînes sont correctement reconnues par Excel. L'objectif de `CONCAT()` dans cet exemple est de tromper ODBC, pour lui faire croire qu'il a à faire avec une colonne de type chaîne. Sans cela, ODBC sait que c'est une colonne de type temps et Excel ne le comprend pas. Notez que c'est un bug d'Excel, car ce dernier converti automatiquement les chaînes en dates. C'est bien lorsque la source est un texte, mais c'est idiot lorsque la source est une connexion ODBC.

odbcadmin

Programme de test pour ODBC.

Delphi

Il faut utiliser DBE 3.2 ou plus récent. Utilisez l'option 'Don't optimize column width' lors de la connexion à ***MySQL***. Par ailleurs, voici un source utile sous delphi, qui prépare la source ODBC et une entrée MyODBC (l'entrée BDE requiert BDE Alias Editor qui peut être disponible librement sur le site Delphi Super Page près de chez vous.) (Merci à Bryan Brunton bryan@flesherfab.com pour celle la).

```
fReg:= TRegistry.Create;
  fReg.OpenKey('\Software\ODBC\ODBC.INI\DocumentsFab', True);
  fReg.WriteString('Database', 'Documents');
  fReg.WriteString('Description', ' ');
  fReg.WriteString('Driver', 'C:\WINNT\System32\myodbc.dll');
  fReg.WriteString('Flag', '1');
  fReg.WriteString('Password', '');
  fReg.WriteString('Port', ' ');
  fReg.WriteString('Server', 'xmark');
  fReg.WriteString('User', 'winuser');
  fReg.OpenKey('\Software\ODBC\ODBC.INI\ODBC Data Sources', True);
  fReg.WriteString('DocumentsFab', 'MySQL');
  fReg.CloseKey;
  fReg.Free;

  Memo1.Lines.Add('DATABASE NAME=');
  Memo1.Lines.Add('USER NAME=');
  Memo1.Lines.Add('ODBC DSN=DocumentsFab');
  Memo1.Lines.Add('OPEN MODE=READ/WRITE');
  Memo1.Lines.Add('BATCH COUNT=200');
  Memo1.Lines.Add('LANGDRIVER=');
  Memo1.Lines.Add('MAX ROWS=-1');
  Memo1.Lines.Add('SCHEMA CACHE DIR=');
  Memo1.Lines.Add('SCHEMA CACHE SIZE=8');
  Memo1.Lines.Add('SCHEMA CACHE TIME=-1');
  Memo1.Lines.Add('SQLPASSTHRU MODE=SHARED AUTOCOMMIT');
  Memo1.Lines.Add('SQLQRYMODE=');
  Memo1.Lines.Add('ENABLE SCHEMA CACHE=FALSE');
  Memo1.Lines.Add('ENABLE BCD=FALSE');
  Memo1.Lines.Add('ROWSET SIZE=20');
  Memo1.Lines.Add('BLOBS TO CACHE=64');
  Memo1.Lines.Add('BLOB SIZE=32');

  AliasEditor.Add('DocumentsFab','MySQL',Memo1.Lines);
```

C++Builder

Testé sous BDE 3.0. Le seul problème connu est lorsque la structure de la table change, les champs de requêtes ne sont pas modifiés. BDE ne semble pas reconnaitre les clés primaires, mais seulement les index primaires, bien que cela n'ai pas généré de problèmes.

Visual basic

Pour pouvoir modifier une table, vous devez définir une clé primaire pour cette table.

16.4 Comment remplir les différents champs du gestionnaire ODBC

Il y a trois possibilités pour spécifier le nom du serveur sous Windows95:

- Utilisez l'adresse IP du serveur.
- Ajouter un fichier `lmhosts' avec les informations suivantes :

```
ip hostname
```

Par exemple :

```
194.216.84.21 my
```

- Configurer le PC pour qu'il utilise le DNS.

Exemple de configuration de ``ODBC setup'':

```
Windows DSN name:   test
Description:        Ceci est ma base de test
MySql Database:     test
Server:             194.216.84.21
User:               monty
Password:           mon_mot_de_passe
Port:
```

`Windows DSN name` peut prendre n'importe valeur qui est unique pour votre configuraiton ODBC.

Vous n'avez pas à spécifier les valeurs de `Server`, `User`, `Password` ou `Port` dans l'écran de configuraiton ODBC. Cependant, si vous le faites, ces valeurs seront ultérieurement utilisées comme valeurs par défaut, lors de la connexion. Vous aurez aussi la possililité de changer ces valeurs lors de la connexion.

Si le numéro de port n'est pas précisé, la valeur par défaut du port est utilisée (33)

Si vous spécifiez l'option `Read options from C:\my.cnf`, les groupes `client` et `odbc` seront lu depuis le fichier `` `C:\my.cnf' ``. Vous pouvez y placer toutes les options de `mysql_options()`. 20.4.37 mysql_options().

16.5 Comment lire la valeur d'une colonne de type `AUTO_INCREMENT` avec ODBC

Un problème classique est de lire la valeur générée lors de la dernière opération d'auto−incrément, lors qu'ne requête `INSERT`. Avec ODBC, vous pouvez toujours faire ceci : (on suppose que votre `auto` est le champs `AUTO_INCREMENT`):

```
INSERT INTO foo (auto,text) VALUES(NULL,'text');
SELECT LAST_INSERT_ID();
```

Ou bien, si vous allez insérer cet ID dans une autre de table, vous pouvez faire ceci :

```
INSERT INTO foo (auto,text) VALUES(NULL,'text');
INSERT INTO foo2 (id,text) VALUES(LAST_INSERT_ID(),'text');
```

Pour utiliser certaines applications ODBC (au moins Delphi et Access), la requête suivante peut servir à retrouver une ligne récemment insérée :

```
SELECT * FROM nom_table WHERE auto IS NULL;
```

17 Utiliser MySQL avec certains programmes

17.1 Utilise MySQL avec Apache

Les programmes proposés dans les contributions vous permettront d'identifier vos utilisateurs à partir d'une base de données ***MySQL*** mais aussi d'enregistrer l'historique dans une base de données ***MySQL*** . http://www.mysql.com/Contrib.

Vous pouvez changer le format d'historique d'Apache pour qu'il soit facilement lisible par MySQL en ajoutant les lignes suivantes dans le fichier de configuration Apache

```
LogFormat \
        "\"%h\",%{%Y%m%d%H%M%S}t,%62;s,\"%b\",\"%{Content-Type}o\",  \
        \"%U\",\"%{Referer}i\",\"%{User-Agent}i\""
```

Avec ***MySQL*** vous pouvez maintenant faire ceci :

```
LOAD DATA INFILE '/local/access_log' INTO TABLE table_name
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"' ESCAPED BY '\\'
```

18 Problèmes et erreurs fréquents

18.1 Que faire si MySQL plante constamment?

Toutes les versions de ***MySQL*** sont testées sur de nombreuses plates−formes avant d'être livrées. Cela n'assure pas qu'il n'y ait aucune erreur dans ***MySQL***, mais si elles existent, elles sont rares et très difficiles à trouver. Si vous avez un problème, il nous sera toujours utile que vous fassiez un bilan complet de votre système, et cela vous permettra aussi de résoudre plus sûrement votre problème.

En premier lieu, il faut savoir si le problème provient de votre démon `mysqld` ou de votre client. Si vous exécutez la commande `mysqladmin version` , vous pourrez savoir le temps de service de votre serveur `mysqld.` Si `mysqld` a terminé son service inopinément, vous en trouverez la raison dans le fichier ``mysql−data−directory/'hostname'.err''.

Etant donné qu'il est très difficile de savoir pourquoi le serveur plante, commencez par vérifier si ce qui plante chez d'autres, plante aussi pour vous. Vous pouvez ainsi tester ce qui suit :

- Arrêtez votre démon `mysqld` avec `mysqladmin shutdown`, puis lancez `isamchk −−silent −−force */*.ISM` sur toutes les tables. Enfin, relancez le démon `mysqld`. Ceci vous assurera que le serveur est en état de marche. Reportez vous à la section `Maintenance.`
- Utilisez `mysqld −−log` pour accéder à l'historique : vous pourrez ainsi savoir si le serveur s'arrête après une requête particulière. Près de 95% des bugs interviennent après la même requête !Généralement, c'est une des dernières requête du fichier d'historique, juste avant les lignes de redémarrage de ***MySQL***. Vous pouvez vérifier ceci avec la séquence suivante :
  - Arrêtez le serveur ***MySQL*** (avec `mysqladmin shutdown`)
  - Make a backup of files in the ***MySQL*** database directory.
  - Vérifiez les toutes tables avec `isamchk −s */*.ISM`. Réparez les tables corrompues avec `isamchk −r chemin−de−la−table.ISM`.
  - Effacez (ou déplacez) les anciens fichiers d'historiques du dossier de donnéesde ***MySQL***.
  - Redémarrez le serveur avec `safe_mysql −−log`.
  - Si `mysqld` s'interromps, vous pourrez exécuter `mysql < mysql−log−file` pour recherche une éventuelle requête rebelle. Vous pouvez aussi mettre le fichier d'historique dans un autre dossier que le dossier par défaut, en lancant ***MySQL*** avec les options `safe_mysqld −−data=path−to−backup−directory`.
- Avez vous essayé les benchmarks ? Elles testent ***MySQL*** dans de nombreux domaines. Vous pouvez aussi ajoutez des lignes de codes qui vont simuler votre application! Les benchmarks sont situées dans le dossier ``bench'' de la distribution, ou, pour les distributions binaire, dans le dossier ``sql−bench'' du dossier d'installation ***MySQL***.
- Essayez `fork_test.pl` et `fork2_test.pl`.
- Vérifiez le fichier ``mysql−data−directory/'hostname'.err'' : il contient peut être les erreurs.
- Si vous configurez ***MySQL*** en mode debuggage, il vous sera plus facile de repérer des erreurs lorsqu'un problème se présentera. Reconfigurez ***MySQL*** avec l'option `−−with−debug` puis recompilez. G.1 Debugguer un serveur MySQL.
- La configuration en mode debug de ***MySQL*** force le mode d'allocation sécurisé : cela met en lumière des erreurs, et fourni un affichage plus important sur le détails des opérations.
- Avez vous appliqué les derniers patchs de votre OS ?
- Utiliser l'option `−−skip−locking` de `mysqld`. Sur certains systèmes, le gestionnaire `lockd` de verrous ne fonctionne par correctement. Cette option (`−−skip−locking`) force `mysqld` à ne pas l'utiliser. (Cela implique que vous ne pourrez pas lancer 2 serveurs `mysqld` sur les mêmes données, et que vous devrez être très prudent lors de l'utilsiation de `isamchk`, mais ce peut être une option instructive.).
- Avez vous essayé la commande `mysqladmin −u root processlist` lorsque `mysqld` semble fonctionner, mais ne pas répondre ? Parfois, `mysqld` est dans le comas, même si il ne semble pas l'être. Le problème peut provenir du fait que toutes les connexions sont prises, ou bien d'un problème interne. `mysqladmin processlist` est toujours utilisable, même en cas de surcharge de connexions, et peut fournir des informations très utiles, comme le nombre de connexions et leur statut.
- Lancez la commande `mysqladmin −i 5 status` dans une nouvelle console, pour avoir des statistiques en temps réel.
- Essayez la combinaison suivante :
  1. Lancez `mysqld` à partir de `gdb` (ou d'un autre debuggeur).
  2. Exécutez vos scripts de tests.
  3. Utilisez la commande `back` (ou backtrace dans votre debuggeur) lorsque le dump de `mysqld` est généré.
- Essayez de simuler votre application avec un script Perl pour forcer ***MySQL*** à crasher ou faire des erreurs.
- Ou enfin, envoyer un rapport d'erreur. 2.3 Comment rapporter des bugs et des problèmes. Mais, soyez encore plus détaillé que d'habitude. Etant donné que ***MySQL*** fonctionne parfaitement pour de nombreuses personnes, le crash peut provenir d'une caractéristique de votre ordinateur. (par exemple, une erreur due à un système de librairies spécifique).
- Si vous avez un problème avec les tables à lignes de taille variable, et que vous n'utilisez pas de colonne de type `BLOB ou TEXT` (et seulement des `VARCHAR`) vous pouvez essayer de remplacer tous les types `VARCHAR` par `CHAR` avec `ALTER TABLE`. Cela va forcer ***MySQL*** à utiliser des lignes à taille fixe. Les lignes à taille fixe prennent un peu plus de place, mais sont beaucoup moins sujette à dégradation. Le gestionnaire de ligne de taille variable a été utilisé à TCX (NDT : qui a concu MySQL) depuis 3 ans au moins sans aucun

problème, mais, par nature, les lignes à taille variables sont plus délicates.

18.2 Erreurs fréquentes avec MySQL

18.2.1 Erreur `MySQL server has gone away`

Cette section couvre aussi l'erreur `Lost connection to server during query`.

La raison la plus probable qui conduit à une erreur `MySQL server has gone away` est que le serveur a dépassé le délai d'attente, et qu'il a terminé la connexion. Par défaut, les connexions sont terminée après 8 heures d'inaction.

Vous pouvez vous assurez qu le serveur ***MySQL*** fonctionne toujours avec la commande `mysqladmin version` et en observant le temps de vie.

Si vous avez un script en cours, vous n'avez qu'à le relancer, et le client se reconnectera automatiquement.

Dans cette situation vous pouvez obtenir les erreurs suivantes (suivant l'OS) :

Vous pouvez voir ces erreurs survenir si vous émettez une erreur trop grande, ou de taille incorrecte. Si `mysqld` recoit un paquet trop grand ou hors service, il suppose alors que quelque chose n'a pas fonctionné du coté du client, et il termine la connexion. Si vous avez des requêtes de grande taille (par exemple, si vous avez des colonnes de type `BLOB`), il vaut mieux augmenter la taille maximale des requête lors du démarrage de `mysqld` avec l'option `−O max_allowed_packet=#` option (par défaut : 1M). La mémoire supplémentaire est allouée à la volée, ce qui permet à `mysqld` de n'utiliser plus de mémoire que lorsque vous émettrez une énorme requête, ou que vous vous attendrez à une énorme réponse.@node Can not connect to server, Blocked host, Gone away, Common errors

18.2.2 Erreur `Can't connect to [local] MySQL server`

Les clients ***MySQL*** peuvent se connecter à un serveur `mysqld` de deux manières différentes : soit les sockets Unix, qui se connecte via un fichier système (par défaut ``/tmp/mysqld.sock''), ou par TCP/IP, qui se connecte à un numéro de port. Les sockets Unix sont beaucoup plus rapides que les sockets TCP/IP mais uniquement sur le même ordinateur. Les sockets Unix sont utilisées par défaut si vous ne spécifiez par de nom d'hôte, ou si vous vous connectez à `localhost`.

L'erreur `Can't connect to ...` signifie normalement qu'il n'y a pas de serveur ***MySQL*** sur le système, ou que vous utilisez un faux fichier de socket Unix ou un mauvais port TCP/IP lorsque vous vous connectez au serveur `mysqld`.

Commencez par vérifier (avec `ps`) qu'il y a bien un processus nommé `mysqld` sur votre serveur! Si il n'y en a pas, il vaut mieux le démarrer. 4.15.2 Problèmes avec le serveur ***MySQL***.

Si un serveur `mysqld` tourne, vous pouvez vérifier le serveur avec les connexions suivantes (le numéro de port ou le nom de fichier de socket Unix peuvent différer de votre système, bien sur...) :

```
shell62; mysqladmin version
shell62; mysqladmin −h `hostname` version
shell62; mysqladmin −h `hostname` −−port=3306 version
shell62; mysqladmin −h 'ip for your host' version
shell62; mysqladmin −−socket=/tmp/mysql.sock version
```

Notez que l'utilisation de guillemets arrière (backquotes) plutôt que des guillemets simples autour de `hostname`; force l'affichage de `hostname` (i.e., l'hôte courant), à utiliser ultérieurement dans la commande `mysqladmin`.

Voici quelques raisons qui causent l'erreur `Can't connect to local MySQL server` :

- `mysqld` n'est pas lancé
- Vous utilisez un système qui utilise MIT−pthreads. Si vous un système qui n'a pas de threads natifs, `mysqld` utilise le package MIT−pthreads. 4.2 Systèmes d'exploitation supportés par MySQL. Cependant, MIT−pthreads ne supporte pas les sockets Unix sockets, ce qui fait que sur ces systèmes, vous devez obligatoirement spécifier un nom d'hôte explicitement, lorsque vous vous connectez à un serveur. Essayez la commande suivante, pour vérifier la connexion au serveur :

```
shell62; mysqladmin −h `hostname` version
```

- Quelqu'un a supprimé le fichier de sockets Unix utilisé par `mysqld` (par défaut ``/tmp/mysqld.sock''). Il est aussi possible que la table de `cron` supprime automatiquement la socket ***MySQL*** (e.g., une t che qui va supprimer les anciens fichiers dans le dossier ``/tmp'' directory). Vous pouvez aussi lancer la commande `mysqladmin version` et vérifier que la socket `mysqladmin` que vous tentez d'utiliser existe vraiment. La solution dans ce cas est de modifier la t che de la table de `cron` pour qu'elle ne supprime par ``mysqld.sock'' ou encore de placer le fichier de socket ailleurs. Vous pouvez ainsi placer le fichier de socket dans un autre dossier avec la commande suivante :

```
shell62; ./configure −−with−unix−socket−path=/path/to/socket
```

Vous pouvez aussi lancer `safe_mysqld` avec l'option `−−socket=/path/to/socket` et assigner une valeur à la variable d'environnement `MYSQL_UNIX_PORT` avant de démarrer votre client ***MySQL***. Puis vous lancez le serveur `mysqld` avec l'option `−−socket=/path/to/socket`. Si vous changez le chemin de la socket sur le serveur, vous devez le notifier aux clients ***MySQL***. Vous pouvez le faire avec la variable d'environnement `MYSQL_UNIX_PORT` ou en fournissant aux clients un chemin jusqu'à la socket. Ainsi, vous pouvez tester la commande suivante :

```
shell62; mysqladmin −−socket=/path/to/socket version
```

- Si vous utilisez Linux et qu'un des thread s'est terminé (core dumped), vous devrez terminer les autres threads `mysqld` (avec le script `mysql_zap`), avant de pouvoir redémarrer un nouveau serveur ***MySQL***. Reportez vous à la section `Crashing `

Si vous avez le message d'erreur `Can't connect to MySQL server on un_nom_d_hote`, vous pouvez essayer les solutions suivantes :

- Verifiez que le serveur est en état de fonctionnement, avec une commande `telnet your−host−name tcp−ip−port−number` et `RETURN` (plusieurs fois). Si il y a un serveur ***MySQL*** sur ce port, vous allez recevoir une réponse qui comportera notamment un numéro de version (celui du serveur en fonctionnement). Si vous recevez une erreur du style : `telnet: Unable to connect to remote host: Connection refused`, C'est qu'il n'y a pas de serveur sur ce port.
- Essayer de vous connecter au serveur depuis la machine locale et vérifiez le port TCP/IP dans la configuration de mysqld (variable `port`) avec la commande `mysqladmin variables`.
- Vérifiez que votre serveur `mysqld` n'a pas été lancé avec l'option `−−skip−networking`.

18.2.3 Erreur `Host '...' is blocked`

Si vous recevez une erreur telle que :

```
Host 'hostname' is blocked because of many connection errors.
Unblock with 'mysqladmin flush−hosts'
```

C'est que `mysqld` a reçu trop de demande de connexion (`max_connect_errors`) de la part de l'hôte `'hostname'` est qu'elles ont été interrompue au cours du processus. Après `max_connect_errors` echecs de connexion, `mysqld` supppose que quelque chose ne va pas (une attaque éventuelle d'un hacker), et bloque toutes les connexions ultérieures, jusqu'à ce que quelqu'un exécute la commande `mysqladmin flush−hosts`.

Par défaut, `mysqld` se bloque après 10 erreurs de connexions. Vous pouvez modifier ce paramètre avec :

```
shell62; safe_mysqld −O max_connect_errors=10000 38;
```

Notez que si vous recevez cette erreur de la part d'un hôte donné, il vaut mieux vérifier qu'il n'y a pas de problème de connexion TCP/IP avec cet hôte.Si les connexions TCP/IP ne fonctionnent pas, augmenter la valeur de `max_connect_errors` n'y changera rien !

18.2.4 Erreur `Out of memory`

Si vous recevez une erreur telle que :

```
mysql: Out of memory at line 42, 'malloc.c'
mysql: needed 8136 byte (8k), memory in use: 12481367 bytes (12189k)
ERROR 2008: MySQL client ran out of memory
```

Cette erreur est une erreur du client `mysql`. Le client manque tout simplement de mémoire pour enregistrer le résultat complet.

Pour remédier au problème, assurez vous que votre requête est correcte, et surtout, retourne t elle un nombre raisonnable de ligne ? Si c'est le cas, vous pouvez utiliser `mysql −−quick`, qui utilise `mysql_use_result()` pour récupérer le résultat, en déplaçant la charge de travail du client vers le serveur.

18.2.5 Erreur `Packet too large`

Lorsqu'un client ***MySQL*** ou le serveur `mysqld` reçoit un paquet de taille supérieure à `max_allowed_packet` octets, il va générer une erreur `Packet too large` et fermer la connexion.

Si vous utilisez le client `mysql`, vous pouvez changer la taille du buffer en lançant le client avec l'option `mysql −−set−variable=max_allowed_packet=8M`.

Si vous utilisez d'autres clients qui ne vous permettent pas de changer cette taille (comme `DBI`), il faut changer la taille du buffer sur le serveur. L'option a modifier est `max_allowed_packet`. Par exemple, démarrer le serveur avec l'option `−−set−variable=max_allowed_packet=24M`.

18.2.6 Erreur `The table is full`

Cette erreur survient lorsqu'une table temporaire excède la taille de `tmp_table_size` octets. Pour éviter ce problème, vous pouvez utiliser l'option `−O tmp_table_size=#` de `mysqld` pour accroître la taille des tables temporaires, ou bien utiliser l'option SQL `SQL_BIG_TABLES` avant d'exécuter la requête. `SET OPTION`.

Vous pouvez aussi démarrer le serveur avec l'option `−−big−tables`. Cela revient à utiliser la clause `SQL_BIG_TABLES` dans toutes les requêtes.

18.2.7 Erreur coté client `Commands out of sync`

L'erreur `Commands out of sync; You can't run this command now` vient du fait que vous appelez les commandes dans le mauvais ordre, dans votre client.

Cela arrive si, par exemple, vous utilisez `mysql_use_result()` puis essayez d'exécuter une nouvelle requête avant d'appeler `mysql_free_result()`. Cela peut aussi arriver si vous exécutez deux requête qui retourne des données, sans employer `mysql_use_result()` ou `mysql_store_result()` entre ces deux commandes.

18.2.8 Erreur `Ignoring user`

Lorsque l'erreur suivante s'affiche :

Found wrong password for user: 'some_user@some_host'; Ignoring user

Cela signifie que lorsque `mysqld` a été démarré, ou qu'il a recharger les tables de permissions, il a trouvé un entrée dans la table `user` avec un mot de passe invalide. Par conséquent, l'entrée est purement et simplement ignorée par le système de droits.

Quelques situations et leur solution :

- Vous avez lancé une nouvelle version de `mysqld` avec une vieille table `user`. Vous pouvez le vérifier en exécutant la commande `mysqlshow mysql user` pour voir si le mot de passe contient moins de 16 caractères. Dans ce cas, corrigez le problème avec le script `scripts/add_long_password`.
- L'utilisateur a un ancien mot de passe de 8 caractères, et vous n'avez pas démarré avec l'option `--old-protocol`. Modifiez l'entrée de l'utilisateur dans la table `user` en lui attribuant un nouveau mot de passe, ou redémarrez le serveur avec l'option `--old-protocol`.
- Vous avez spécifié un mot de passe dans la table `user` sans utiliser la fonction `PASSWORD()`. Utilisez le client `mysql` pour modifier l'entrée de l'utilisateur dans la table `user` , et attribuez lui un nouveau mot de passe. Assurez vous que vous utilisez bien la fonction `PASSWORD()`:

```
mysql62; update user set password=PASSWORD('your password')
           where user='XXX';
```

18.2.9 Erreur `Table 'xxx' doesn't exist`

L'erreur `Table 'xxx' doesn't exist` ou `Can't find file: 'xxx' (errno: 2)` signifie que la table demandée n'existe pas dans la bas de données courante.

N'oubliez pas que ***MySQL*** utilise des dossiers et des fichiers pour enregistrer ses bases de données et tables, et que les noms de ces derniers sont ***sensibles à la casse.*** (Sous Win32 les noms des bases de données et tables ne sont pas sensibles à la casse mais toutes les références à une table donnée dans une même requête doivent avoir la même casse ! !).

Vous pouvez lister les tables disponibles avec la requête `SHOW TABLES`. Reportez vous à la section `SHOW`.

18.3 Comment MySQL gère les disques pleins

Lorsqu'un disque est plein, ***MySQL*** effectue les opérations suivantes :

- Il vérifie toutes les minutes si suffisamment de place a été libérée pour pouvoir écrire les lignes courantes. Si oui, ces lignes sont écrites, si non, il attend une nouvelle minute.
- Toutes les 6 minutes, il ajoute une ligne dans le fichier d'historique.

Pour résoudre ce problème, vous pouvez essayer ce qui suit :

- Pour continuer, il suffit de libérer suffisament d'espace sur le disque pour que toutes les lignes puissent être écrites.
- Pour abandonner le thread, vous devez envoyez une commande `mysqladmin kill`. Le thread sera terminé lors de sa prochaine tentative d'écriture. (au pire, dans une minute).
- N'oubliez pas que d'autres threads risquent d'être mis en attente, à cause de cette erreur ``disk full''. Si vous avez plusieurs threads, terminer celui qui était bloqué par la capacité du disque, peut permettre de déverrouiller une table, et autoriser d'autres thread à continuer. Cependant, les prochains threads en écriture seront aussi bloqués.

18.4 Comment exécuter des commandes SQL depuis un fichier text

Le client `mysql` est typiquement utilisé de manière interactive de la manière suivante :

```
shell62; mysql database
```

Cependant, il est possible de mettre un jeu de commande dans un fichier, et d'indiquer à `mysql` de lire les commandes depuis ce fichier. Pour cela, créez un fichier texte ``fichier_texte'' qui contient les commandes que vous souhaitez exécuter. Puis, lancez `mysql` comme ceci :

```
shell62; mysql database 60; fichier_texte
```

Vous pouvez aussi commencer votre fichier texte avec la commande `USE nom_base_de_donnees`. Dans ce cas, il n'est pas nécessaire de spécifier le nom de la base de donnée de travail dans la ligne de commande.

```
shell62; mysql 60; text_file
```

12.1 Présentation des différents programmes MySQL.

18.5 Où MySQL enregistre les fichiers temporaires

MySQL utilise la variable d'environnement `TMPDIR` comme chemin d'accès au dossier de fichiers temporaires. Si vous n'avez pas un tel dossier, ***MySQL*** utilise par défaut celui du système, c'est à dire généralement ``/tmp'' ou ``/usr/tmp''. Si le dossier de fichier temporaire est trop petit, il vaut mieux le changer vers un autre dossier qui soit suffisamment grand, en modifiant la valeur de `safe_mysqld`! Vous pouvez aussi utiliser l'option de démarrage `--tmpdir`.

MySQL crée ses fichiers temporaires en tant que fichiers cachés. Cela garanti que les fichiers ne seront effacés que lorsque `mysqld` sera arrêté. L'inconvénient de cette méthode est que vous ne pourrez pas repérer le fichier temporaire gigantesque qui remplit votre dossier temporaire !

Lorsque vous utilisez des tris (`ORDER BY` ou `GROUP BY`), ***MySQL*** utilise généralement un ou deux fichiers temporaires. L'espace disque nécessaire est :

```
(longueur des données triées + taille de (pointeur de base de donnée))
* nombre de ligne utilisées
* 2
```

taille de (pointeur de base de donnée) vaut généralement 4, mais risque d'être plus grand encore, lorsqu'il faudra gérer des tables vraiment grandes.

Pour certaines requêtes SELECT, ***MySQL*** crée aussi une table temporaire. Celles ci ne sont pas cachés, et leur noms commence par ``SQL_*''.

ALTER TABLE et OPTIMIZE TABLE crée toutes les deux des tables temporaires, mais dans le dossier d'origine.

18.6 Comment protéger `/tmp/mysql.sock' contre l'effacement?

Si vous avez des problèmes parce que n'importe qui peut effacer le fichier de socket ``/tmp/mysql.sock'', vous pouvez, sur la plus part des versions d'UNIX, protéger le dossier ``/tmp'' en lui attribuant le sticky bit. Connectez vous comme root et faites ceci :

```
shell62; chmod +s /tmp
```

Cela protège votre dossier ``/tmp'' et les fichiers qui sont dedans ne pourront être effacés que par leur possesseurs (root).

Vous pouvez vérifier que le sticky bit est en place avec ls −ld /tmp. Si la dernière permission affichée est t, le sticky bit est en place.

18.7 Erreur Access denied

6.6 Fonctionnement du système de droits. Et particulièrement 6.13 Causes des erreurs "Access denied".

18.8 Comment faire tourner MySQL en tant qu'utilisateur normal

Un serveur ***MySQL*** peut être lancé par n'importe quel utilisateur. Pour modifier mysqld et le faire fonctionner avec l'utilisateur nom_utilisateur, vous devez faire ceci :

1. Stoppez l serveur, si il fonctionnait (mysqladmin shutdown).
2. Modifiez les dossiers et fichiers de la base pour que nom_utilisateur ait les droits de lecture et écriture sur ces fichiers. (Il vous faudra peut être un accès root):
   ```
   shell62; chown −R user_name /path/to/mysql/datadir
   ```
   Si les dossiers ou fichiers dans le dossier de données de ***MySQL*** sont des liens symboliques, vous aurez aussi à suivre ces liens, et donner les droits d'accès sur les dossiers et fichiers pointés. . chown −R ne suivra pas les liens pour vous.
3. Lancez le serveur en tant que nom_utilisateur, ou, si vous utilisez la version ***MySQL*** 3.22 ou plus récent, lancez mysqld en tant que root Unix et utilisez l'option −−user=user_name. mysqld effectuera le changement d'utilisateur avant d'accepter les connexions.
4. Si vous utilisez le script mysql.server pour démarrer mysqld lorsque le système redémarre, vous aurez à le modifier pour que mysqld démarrer bien en tant que nom_utilisateur, ou lancez mysqld avec l'option −−user option. (Aucune modification de safe_mysqld n'est nécessaire).

A partir de ce moment, mysqld doit fonctionner sans problèmes, avec l'utilisateur nom_utilisateur . Cependant, une chose n'a pas changé, ce sont les permissions sur les tables. Par défaut, juste après avoir exécuter le script mysql_install_db), l'utilisateur root est le seul a avoir des autorisations d'accès, de création de bases. A moins que vous n'ayez modifié ces permissions, elles ont toujours cours. Cela ne va pas vous empêcher d'accéder à l'utilisateur ***MySQL*** root lorsque vous êtes connecté en tant qu'utilisateur Unix nom_utilisateur : pensez à spécifier l'option −u root dans le client.

Remarquez bien que accéder à ***MySQL*** en tant que `root`, avec l'option `−u root` de la ligne de commande n'a rien à voir avec le fait que ***MySQL*** est lancé par le `root` Unix, ou n'importe quel autre utilisateur Unix. Les conditions d'accès et les noms d'utilisateurs de ***MySQL*** sont complétement différents de ceux Unix. La seule liaison qu'il y a entre les deux est que si vous essayez de vous connecter à la base ***MySQL*** sans fournir de nom d'utilisateur, le client va tenter de se connecter en utilisant votre nom d'utilisateur Unix comme nom d'utilisateur ***MySQL***.

Si votre système Unix n'est pas sécurisé, vous devrez au moins mettre un mot de passe pour l'utilisateur `root` ***MySQL***. Sinon, n'importe quelle personne avec un compte sur cette machine peut utiliser la commande `mysql −u root nom_base_de_donnees` et faire ce qu'elle veut.

18.9 Problèmes avec les permissions fichiers

Si vous avez des problèmes avec les droits d'accès, par exemple, si vous `mysql` retourne l'erreur suivante lors de la création d'une table :

```
ERROR: Can't find file: 'path/with/filename.frm' (Errcode: 13)
```

C'est que la variable d'environnement `UMASK` a été incorrectement affectée lors du démarrage de `mysqld`. La valeur par défaut de umask est `0660`. Vous pouvez la modifier avec :

```
shell62; UMASK=384  # = 600 in octal
shell62; export UMASK
shell62; /path/to/safe_mysqld 38;
```

18.10 File not found

Les erreurs `ERROR '...' not found (errno: 23)`, `Can't open file: ... (errno: 24)` ou n'importe quelle erreur avec `errno 23` ou `errno 24`, signifie que vous n'avez pas alloué assez de pointeurs de fichiers pour ***MySQL***. Vous pouvez utiliser l'utilitaire `perror` pour avoir une description de la signification de l'erreur :

```
shell62; perror 23
File table overflow
shell62; perror 24
Too many open files
```

Le problème est que `mysqld` tente de garder ouvert trop de fichiers simultanément. Vous pouvez limiter le nombre de fichier ouvert simultanément, ou bien augmenter le nombre autorisé de fichier ouverts.

Pour réduire le nombre de fichiers ouverts par `mysqld`, réduisez la taille du fichier de cache des tables, avec l'option `−O table_cache=32` de `safe_mysqld` (la valeur par défaut est de 64). Réduire la valeur de `max_connections` réduira aussi le nombre de fichiers ouverts (par défaut, 90).

Pour changer le nombre maximum de pointeurs de fichiers simultané dans `mysqld`, modifiez le script `safe_mysqld`. Il y a une ligne du script mise en commentaire et qui vaut : −out line `ulimit −n 256`. Vous pouvez supprimer le caractère de commentaire `'#'`, et changer la valeur par défaut de 256 par la valeur que vous souhaitez.

ulimit permet d'augmenter le nombre de pointeur de fichiers, mais sa valeur sera limité par le système d'exploitation. Si vous donnez une valeur qui est supérieure à ce que votre système d'exploitation peut

supporter, elle sera inefficace. Consultez la documentation de votre système d'exploitation pour connaître cette valeur.

Notez que si vous lancez un shell `tcsh`, `ulimit` ne fonctionnera pas! `tcsh` affichera aussi les valeurs incorrectes si vous demandez les valeurs courantes! Dans ce cas, il vaut mieux démarrer `safe_mysqld` avec `sh`!

18.11 Problèmes avec les colonnes DATE

Le format d'une colonne de type `DATE` est `'YYYY−MM−DD'`. Suivant la norme ANSI SQL, aucun autre format n'est toléré. Il vous faut utiliser ce format dans les expressions `UPDATE` et dans les clauses WHERE (`SELECT`). Par exemple:

```
mysql62; SELECT * FROM nom_table WHERE date 62;= '1997−05−05';
```

De manière pratique, ***MySQL*** converti automatiquement une date en nombre si la date est utilisée dans un contexte numérique (et vice versa). Il est aussi capable d'utiliser une forme non conventionnelle lors des modification, ou dans les clauses `WHERE` lors de comparaisons avec des colonnes de type `TIMESTAMP`, `DATE` ou `DATETIME`. (Non conventionnelle signifie ici que des caractères non numériques peuvent être utilisé ici. Par exemple, `'1998−08−15'` et `'1998#08#15'` sont équivalents.) ***MySQL*** peut aussi convertir une chaîne ne contenant pas de séparateur en date, en supposant que cela puisse avoir un sens, comme `'19980815'`.

La date particulière de `'0000−00−00'` peut être enregistrée et lue sous la forme de `'0000−00−00'.` Lorsque vous utilisez une date telle que `'0000−00−00'` avec ***MyODBC***, elle sera automatiquement converti en `NULL` par ***MyODBC*** 2.50.12 et plus récent, car ODBC ne peut pas gérer ce genre de date.

Etant donné que ***MySQL*** effectue les conversions précédentes, les exemples suivant fonctionnent :

```
mysql62; INSERT INTO nom_table (idate) VALUES (19970505);
mysql62; INSERT INTO nom_table (idate) VALUES ('19970505');
mysql62; INSERT INTO nom_table (idate) VALUES ('97−05−05');
mysql62; INSERT INTO nom_table (idate) VALUES ('1997.05.05');
mysql62; INSERT INTO nom_table (idate) VALUES ('1997 05 05');
mysql62; INSERT INTO nom_table (idate) VALUES ('0000−00−00');
mysql62; SELECT idate FROM nom_table WHERE idate 62;= '1997−05−05';
mysql62; SELECT idate FROM nom_table WHERE idate 62;= 19970505;
mysql62; SELECT mod(idate,100) FROM nom_table WHERE idate 62;= 19970505;
mysql62; SELECT idate FROM nom_table WHERE idate 62;= '19970505';
```

However, the following will not work:

```
mysql62; SELECT idate FROM nom_table WHERE STRCMP(idate,'19970505')=0;
```

STRCMP() est une fonction de chaîne de caractère, qui va convertir `idate` en chaîne et effectuer une comparaison de type chaîne. to a string. Par contre, il ne va pas convertir `'19970505'` en date, et effectuer la comparaison.

Notez aussi que ***MySQL*** ne s'assure pas qu'une date est correcte ou pas. Si vous enregistrez une date incorrecte telle que `'1998−2−31'`, cette date sera enregistrée. Si une date ne peut pas être converti raisonnablement, elle sera remplacée par `0` lors de l'enregistrement. Cette technique permet d'accélerer les traitement, et nous pensons que c'est à l'application de vérifier les dates, et non pas au serveur.

18.12 Problèmes de décalage horaire

Si vous avez des problèmes avec `SELECT NOW()` qui vous retournerai des valeurs au format GMT et non pas dans votre heure locale, c'est que vous n'avez pas mis la variable d'environnement `TZ`, à la bonne valeur. Il faut l'avoir correctement paramétré avant de lancer le serveur, ou lors de son lancement, avec `safe_mysqld` ou `mysql.server`.

18.13 Sensibilité à la casse dans les recherches

Par défaut, ***MySQL*** effectue des recherche qui sont insensibles à la casse (bien qu'il existe des caractères qui soient insensible à la casse, tel que `czech`. Cela signifie que si vous analysez une colonne avec la formule `nom_colonne LIKE 'a%'`, vous allez trouver toutes les lignes qui commence par `A` ou `a`. Si vous voulez rendre la recherche sensible à la casse, utilisez de préférence `INDEX(nom_colonne, "A")=0` pour analyser un préfixe. Ou bien utilisez `STRCMP(nom_colonne, "A") = 0` , si la colonne doit être exactement un `"A"`.

Les opérateurs de comparaison simples (`>=, >, = , < , <=`, tri et regroupement) sont basé sur la valeur de tri de chaque caractère. Les caractères avec la même valeur de tri (comme E, e et 'e) sont traités de la même façon.

Les comparaisons `LIKE` sont faites avec les valeurs majuscules de chaque caractère (E == e mais E <> 'e)

Si vous voulez qu'une colonne soit toujours traitée en tenant compte de la casse, appliquez lui l'attribut `BINARY`. Voir aussi la section `CREATE TABLE`.

Si vous utilisez des données en Chinois, avec l'encodage big5, il est préférable de donner à toutes vos colonnes l'attribut de `BINARY`. Cela fonctionnera, car le tri des caractères big5 est basé sur l'ordre ASCII.

18.14 Problèmes avec la valeur `NULL`

Le concept de valeur `NULL` est une source intarrissable d'erreur pour les néophytes de SQL, qui pensent souvent que `NULL` est équivalent à la chaîne vide `''`. Ceci est totalement faux! Par exemple, les deux commandes suivantes sont complètement différentes :

```
mysql62; INSERT INTO my_table (phone) VALUES (NULL);
mysql62; INSERT INTO my_table (phone) VALUES ("");
```

Les deux exemples insèrent une valeur dans la colonne `phone`, mais le premier insère la valeur `NULL` et le second insère la chaîne vide. La première commande peut être considéré comme ``numéro de téléphone inconnu'' et le deuxième signifiant : ``elle n'a pas de téléphone''.

En SQL, la valeur `NULL` est toujours fausse lors des comparaisons avec une autre valeur, même avec elle même. Une expression qui contient `NULL` retournera toujours `NULL` à moins que cela ne soit indiqué dans la documentation des opérateurs et fonction mises en jeu. Toutes les colonnes suivantes retourneront `NULL`:

```
mysql62; SELECT NULL,1+NULL,CONCAT('Invisible',NULL);
```

Si vous voulez rechercher des colonnes qui contiennent des valeurs `NULL`, vous ne pourrez pas utiliser le test

=NULL. La commande suivant retournera toujours 0 lignes, car `expr = NULL` est TOUJOURS fausse, pour toute expression:

```
mysql62; SELECT * FROM my_table WHERE phone = NULL;
```

Pour rechercher des valeurs `NULL`, vous devez utiliser le test `IS NULL`. Les exemples suivants montrent comment rechercher les valeurs `NULL` dans la colonne du numéro de téléphone :

```
mysql62; SELECT * FROM my_table WHERE phone IS NULL;
mysql62; SELECT * FROM my_table WHERE phone = "";
```

Avec ***MySQL***, comme pour de nombreux serveurs SQL, vous ne pouvez pas indexer des colonnes qui contiennent des valeurs `NULL`. Vous devez impérativement déclarer les colonnes comme `NOT NULL`. Par conséquent, il ne sera pas possible d'insérer une valeur `NULL` dans une colonne indexée.

Lorsque vous chargez une table avec `LOAD DATA INFILE`, les colonnes vides seront remplies par `''`. Si vous voulez insérer une valeur `NULL` dans une colonne, vous devrez utiliser la séquence `\N` dans le fichier texte. Le mot `'NULL'` ne peut être utilisé que dans certaines circonstances. Reportez vous à la section LOAD DATA.

Lorsque vous utiliser une clause `ORDER BY`, les valeurs `NULL` apparaîtront en premier. Si vous triez dans l'ordre descendant (`DESC`), `NULL` seront placées en dernier. Avec les clauses `GROUP BY`, toutes les valeurs `NULL` sont considérées comme égales.

Pour faciliter la manipulation des valeurs `NULL`, vous disposez des opérateurs `IS NULL` et `IS NOT NULL` et de la fonction `IFNULL()`.

Pour certains types de colonnes, `NULL` est géré de manière particulière. Si vous insérez `NULL` dans la première colonne de type `TIMESTAMP` d'une table, la date courante sera insérée. Si vous insérez `NULL` dans une colonne `AUTO_INCREMENT`, le numéro suivant sera inséré.

18.15 Problèmes avec `alias`

Vous pouvez utiliser les alias pour faire référence à une colonne dans les clauses `GROUP BY`, `ORDER BY` ou `HAVING`. Les alias peuvent aussi servir à donner d'autres noms à certaines colonnes

```
SELECT SQRT(a*b) as rt FROM table_name GROUP BY rt HAVING rt 62; 0;
SELECT id,COUNT(*) AS cnt FROM table_name GROUP BY id HAVING cnt 62; 0;
SELECT id AS "Customer identity" FROM table_name;
```

N'oubliez pas que la norme ANSI SQL ne vous autorise pas à utiliser un alias dans les clauses `WHERE`. En effet, dans la clause `WHERE` les valeurs des colonnes n'ont pas encore été affectée. Par exemple, la commande suivante n'est pas valide :

```
SELECT id,COUNT(*) AS cnt FROM table_name WHERE cnt 62; 0 GROUP BY id;
```

La commande `WHERE` est exécutée pour déterminer quelle ligne doit être inclus dans la clause `GROUP BY` tandis que `HAVING` sert à reconnaître les lignes du résultat qui seront utilisées.

18.16 Effacer des lignes dans les tables liées

Etant donné que ***MySQL*** ne supporte ni les sub−selects ni les effacement sur des tables multiples, vous devrez utiliser l'approche suivante pour effacer des lignes dans deux tables en même temps

1. Sélectionner les lignes dans la table principale, avec une clause `WHERE`.
2. Effacer ces lignes avec les mêmes conditions.
3. `DELETE FROM related_table WHERE related_column IN (selected_rows)`

Si le nombre total de caractères dans la requête avec `related_column` dépasse 1,048,576 (la valeur par défaut de `max_allowed_packet`), il vous faudra scinder la requête en plusieurs sous−requête de plus petite taille, et exécuter des `DELETE` multiples. Le meilleur rendement se situe dans une fourchette de 100−1000 effacements dans `related_column` à chaque fois, si `related_column` est un index. Si `related_column` n'est pas un index, la vitesse sera indépendante du nombre d'arguments de la clause `IN`.

18.17 Résoudre les problèmes sans lignes correspondantes

Si vous avez des requêtes longues et compliquées, avec de nombreuses tables, et qui ne renvoie aucune ligne, vous pouvez utiliser la méthode suivante pour trouver l'erreur dans votre requête :

1. Lancez la requête avec `EXPLAIN` est assurez vous que tout est bon. Reportez vous à la section `EXPLAIN`.
2. Sélectionnez seulement les champs qui sont utilisé dans la clause `WHERE`.
3. Supprimez une table par une table, dans la requête, jusqu'à ce qu'elle vous retourne enfin des lignes. Si les tables sont grandes, c'est une bonne idée d'utiliser la clause `LIMIT 10`.
4. Effectuer un `SELECT` dans la colonne qui devrait retourner des lignes, et comparez la avec la dernière table qui a été supprimé de la requête.
5. Si vous faites des comparaisons entre des `FLOAT` ou `DOUBLE` et des nombres à virgules, vous ne pourrez pas utiliser l'opérateur =! Ce problème est commun à la plus part des langages informatiques, car la représentation en virgule flottante n'est pas une valeur exacte.
   ```
   mysql62; SELECT * FROM table_name WHERE float_column=3.5;
      −62;
   mysql62; SELECT * FROM table_name WHERE float_column between 3.45 and 3.55;
   ```
   Dans la plus part des cas, changer le `FLOAT` en `DOUBLE` résoudra le problème !
6. Si vous ne savez toujours pas ce qui est faux, créez un texte minimum, qui pourra être exécuté avec `mysql test < query.sql`. Vous pourrez créer un fichier de teste avec `mysqldump −−quick database tables > query.sql`. Editez le fichier, supprimez quelques lignes (certaines sont en trop), et ajoutez votre commande à la in. Vérifier que vous avez toujours votre problème avec :

```
shell62; mysqladmin create test2
shell62; mysql test2 60; query.sql
```

Envoyez votre test avec `mysqlbug` à mysql@lists.mysql.com.

18.18 Problèmes avec ALTER TABLE.

Si `ALTER TABLE` s'interromps avec une erreur telle que:

```
Error on rename of './database/name.frm' to './database/B−a.frm' (Errcode: 17)
```

Le problème est que ***MySQL*** a planté durant la dernière commande `ALTER TABLE` et qu'il y a une ancienne table du nom de ``A−quelquechose'' ou ``B−quelquechose''. Dans ce cas, allez dans le dossier de données ***MySQL*** et effacez toues les fichiers dont les noms commencent par A− ou B−. (Ou mieux, déplacez les simplement dans un autre dossier).

ALTER TABLE fonctionne de la manière suivante :

- Il crée une nouvelle table avec le nom ``A−xxx'' contenant les modifications demandées.
- Touts les lignes sont copiées de la table initiale dans la table ``A−xxx''.
- La vieille table est renommée ``B−xxx''.
- `A−xxx' est renommée, et prend le nom de la vieille table
- `B−xxx' est effacé.

Si quelques chose ne fonctionne par durant ce processus, ***MySQL*** tente d'annuler les modifications. Si quelque chose d'important survient, ***MySQL*** risque de laisser la vieille table ``B−xxx'' et un simple renommage peut vous rendre vos données.

18.19 Comment changer l'ordre des colonnes dans une table

Un des points forts SQL est de dissocier l'application du format de stockage. Il est toujours possible de spécifier l'ordre dans lequel vous souhaitez lire les informations.

```
SELECT nom_colonne1, nom_colonne2, nom_colonne3 FROM nom_table;
```

Retournera les colonnes dans l'ordre order `nom_colonne1`, `nom_colonne2`, `nom_colonne3`, tandis que:

```
SELECT nom_colonne1, nom_colonne3, nom_colonne2 FROM nom_table;
```

les retournera dans l'ordre `nom_colonne1`, `nom_colonne3`, `nom_colonne2`.

Il est préférable de ne ***JAMAIS*** lire les colonnes en fonction de leur position, surtout dans un `SELECT *`, car l'ordre des colonnes ne peut jamais être garanti. Il suffit d'une modification dans la base de données pour que votre application n'ait plus de sens.

Pour changer l'ordre des colonnes, vous pouvez suivre la méthode suivante :

1. Créer une nouvelle table, avec l'ordre des colonnes désiré
2. Exécutez `INSERT INTO new_table SELECT fields−in−new_table−order FROM old_table`.
3. Effacez ou renommez `old_table`
4. `ALTER TABLE new_table RENAME old_table`

19 Résolution de problèmes courants avec MySQL

19.1 Réplication de base de données

Pour répliquer une base de données, le moyen courant est l'utilisation de l'historique de modification. 9.2 Historique de modification. Cela impose à la base dont les données change d'agir en tant que maître, et des autres bases seront ses clients. Pour modifier une base cliente, il suffit de lancer l'utilitaire `mysql < update_log`, avec les informations d'hôte, utilisateur et mot de passe.pour accéder à la base cliente, et d'utiliser la base maître comme source.

Si vous n'avez jamais effacé quoique ce soit d'une table, vous pouvez utiliser une colonne de type `TIMESTAMP` pour savoir quelle colonne a été insérée pour modifiée dans une table depuis la dernière réplication (en comparant les dates de la dernière réplication) : vous pouvez alors ne copier que les nouvelles lignes dans la table.

Il est possible de faire une modification full duplex, en utilisant l'historique de modification (pour les effacements), et les timestamps (pour le reste). Mais, dans ce cas, vous devez être capable de gérer les conflits de données, si les valeurs ont été modifiées dans les deux tables en même temps. Vous souhaiterez probablement garder la vieille version pour pouvoir choisir laquelle est la bonne.

Etant donné que la réplication se fait avec des commandes SQL, vous ne pouvez pas utiliser les fonctions suivantes dans les requêtes lorsque vous modifiez les base de données, car elle risque de ne pas retourner la même valeur que dans la base originale :

- `DATABASE()`
- `GET_LOCK()` and `RELEASE_LOCK()`
- `RAND()`
- `USER()`, `SYSTEM_USER()` or `SESSION_USER()`
- `VERSION()`

Toutes les fonctions de temps sont valides, car un timestamp est envoyé au miroir en cas de besoin.. `LAST_INSERT_ID()` est aussi valide.

19.2 Sauvegarde de base de données

Etant donné que les tables de ***MySQL*** sont enregistrées sous la forme de fichiers, les sauvegardes sont très faciles à faire. Pour créer une sauvegarde cohérente, verrouillez les tables dont vous avez besoin avec `LOCK TABLES`. `LOCK TABLES`. Vous n'aurez besoin que d'un verrou de lecture, cequi permettra aux autres threads de continuer à interroger la base, tout en vous laissant le champs libre pour faire la copie des fichiers de la base. Si vous voulez faire une requête SQL pour sauver la table, utilisez donc `SELECT INTO OUTFILE`.

Un autre moyen de faire des sauvegardes d'une base de données est d'utiliser l'utilitaire `mysqldump` :

1. Sauvegarde complète des bases de données :
```
shell62; mysqldump --tab=/path/to/some/dir --lock-tables --opt
```

   Vous pouvez aussi simplement faire une copie de touts les fichiers de tables (``*.frm'', ``*.ISD'' et ``*.ISM''), tant que le

serveur ne fait pas de modiication.

2. Stoppez le serveur `mysqld` si il était lancé, puis relancez le avec l'option `−−log−update`. Vous allez obtenir des fichiers d'historique avec des noms du type ``nomDHote.n'', où `n` est un nombre incrémenté à chaque fois que vous faites un `mysqladmin refresh` ou `mysqladmin flush−logs`, une commande `FLUSH LOGS` statement, ou que vous redémarrez le serveur. Ces fichiers d'historiques fournissent les informations dont vous aurez besoin pour répliquer une base, ou la régénérer à partir de la sauvegarde faite avec `mysqldump`.

Si vous devez reconstruire une base ou une table, essayer d'abord de récupérer vos tables avec `isamchk −r`. Cet utilitaire fonctionne dans 99.9% des cas. Si `isamchk` échoue, utilisez la procédure suivante :

1. Réinstallez la sauvegarde originale à partir de `mysqldump`.
2. Lancez la commande suivante pour re excécuter les modifications à partir des fichiers d'historique

```
shell62; ls −1 −t −r hostname.[0−9]* | xargs cat | mysql
```

ls est utilisé pour mettre les fichiers d'historiques dans le bon ordre. Vous pouvez aussi effectuer des sauvegardes sélectives avec `SELECT * INTO OUTFILE 'nom_de_fichier' FROM nom_table` et les recharger avec `LOAD DATA INFILE 'nom_de_fichier' REPLACE ...` Pour éviter d'avoir des enregistrements en double, utilisez une colonne de type `PRIMARY KEY` ou `UNIQUE` dans la table. L'option `REPLACE` forcera le remplacement des anciens enregistrements par les nouveaux, dès que l'on tentera de doubler une valeur de la clé unique.

19.3 Faire tourner plusieurs serveurs MySQL sur la même machine

Il existe des cas où vous aurez besoin d'avoir plusieurs serveur MySQL sur la même machine. Par exemple, si vous voulez tester une nouvelle version sans perturber le serveur de production. Ou encore, si vous voulez fournir un serveur MySQL pour différents clients.

Pour ce faire, le moyen le plus simple et de compiler le serveur avec différents ports TCP/IP et différentes sockets, pour éviter que les serveurs utilisent les mêmes sockets ou ports.

Supposons qu'il existe un serveur configuré avec le port et la socket par défaut. Alors, pour réaliser une nouvelle configuration, vous pouvez utiliser la commande suivante :

```
shell62; ./configure  −−with−tcp−port=port_number \
             −−with−unix−socket=file_name \
             −−prefix=/usr/local/mysql−3.22.9
```

Ici, `port_number` et `file_name` doivent prendre des valeurs différentes des valeurs par défaut. L'should be different than the default port number and socket file pathname, et la valeur de `−−prefix` doit mener à une autre installation d'un ***MySQL***.

Vous pouvez connaître la socket et le port utilisé actuellement par ***MySQL*** avec la commande suivante :

```
shell62; mysqladmin −h hostname −−port=port_number variables
```

Si un serveur ***MySQL*** utilisait le port que vous avez utilisé, vous allez voir s'afficher la liste des variables de configuration les plus importantes de ce serveur, y compris le nom de la socket.

Il vous faut aussi éditer le script d'initialisation de votre machine (probablement ``mysql.server'') pour démarrer et arrêter plusieurs serveurs `mysqld`.

Vous n'avez pas besoin de recompiler un nouveau serveur ***MySQL*** pour changer ses port et socket. Vous pouvez les changer en les spécifiant lors du démarrage, avec l'option `safe_mysqld`:

```
shell62; /path/to/safe_mysqld --socket=file_name --port=port_number
```

Si vous voulez faire fonctionner le nouveau serveur MySQL avec les mêmes bases de données que l'autre, vous devrez aussi spécifier les fichiers d'historique dans `safe_mysqld` avec mes options `--log` et `--log-update`. Sinon, les serveurs vont être en concurrence sur les mêmes fichiers d'historique .

Attention: Normalement, vous ne devriez jamais avoir deux serveurs qui modifient en même temps des données dans une base. Si votre OS ne supporte pas le verrouillage en écriture, vous allez rencontrer quelques surprises déplaisantes.

Si vous voulez utiliser un autre dossier pour les bases du second serveur, vous pouvez utiliser l'option `--datadir=path` de `safe_mysqld`.

Lorsque vous voulez vous connecter à un serveur ***MySQL*** qui fonctionne sur un port différent du port par défaut, vous pouvez utiliser une des méthodes suivantes :

- Démarrez le client avec l'option `--host 'hostname' --port=port_numer` ou `[--host localhost] --socket=file_name`.
- Avec les programmes en C ou Perl, vous pouvez préciser l'adresse IP et le port lors de la connexion.
- Changez les valeurs des variables d'environnement `MYSQL_UNIX_PORT` et `MYSQL_TCP_PORT` pour pointer sur les sockets et ports par défaut. Si vous voulez utiliser un port ou une socket spécifique, il vaut mieux les placer dans le fichier ``.login''`. 12.1 Présentation des différents programmes MySQL.
- Spécifiez les ports et sockets par défaut dans le fichier ``.my.cnf''` de votre dossier racine. 4.15.4 Fichier d'options.

20 API MySQL

20.1 MySQL C API

The C API code is distributed with ***MySQL***. It is included in the `mysqlclient` library and allows C programs to access a database.

Many of the clients in the MySQL source distribution are written in C. If you are looking for examples that demonstrate how to use the C API, take a look at these clients.

Most of the other client APIs (all except Java) use the `mysqlclient` library to communicate with the ***MySQL*** server. This means that, for example, you can take advantage of many of the same environment variables that are used by other client programs, because they are referenced from the library. See 12.1 Présentation des différents programmes MySQL, for a list of these variables.

The client has a maximum communication buffer size. The size of the buffer that is allocated initially (16K bytes) is automatically increased up to the maximum size (the default maximum is 24M). Since buffer sizes are increased only as demand warrants, simply increasing the default maximum limit does not in inself cause more resources to be used. This size check is mostly a check for erroneous requêtesand communication packets.

The communication buffer must be large enough to contain a single SQL statement (for client−to−server traffic) and one row of returned data (for server−to−client traffic). Each thread's communication buffer is dynamically enlarged to handle any query or row up to the maximum limit. For example, if you have `BLOB` values that contain up to 16M of data, you must have a communication buffer limit of at least 16M (in both server and client). The client's default maximum is 24M, but the default maximum in the server is 1M. You can increase this by changing the value of the `max_allowed_packet` parameter when the server is started. 10.1 Optimisation des valeurs du serveur.

The ***MySQL*** server shrinks each communication buffer to `net_buffer_length` bytes after each query. For clients, the size of the buffer associated with a connection is not decreased until the connection is closed, at which time client memory is reclaimed.

20.2 Types de données C API

`MYSQL`
: This structure represents a handle to one database connection. It is used for almost all ***MySQL*** functions.

`MYSQL_RES`
: This structure represents the result of a query that returns rows (`SELECT`, `SHOW`, `DESCRIBE`, `EXPLAIN`). The information returned from a query is called the *result set* in the remainder of this section.

`MYSQL_ROW`
: This is a type−safe representation of one row of data. It is currently implemented as an array of counted byte strings. (You cannot treat these as null−terminated strings if field values may contain binary data, because such values may contain null bytes internally.) Rows are obtained by calling `mysql_fetch_row()`.

`MYSQL_FIELD`
: This structure contains information about a field, such as the field's name, type and size. Its members are described in more detail below. You may obtain the `MYSQL_FIELD` structures for each field by calling `mysql_fetch_field()` repeatedly. Field values are not part of this structure; they are contained in a `MYSQL_ROW` structure.

`MYSQL_FIELD_OFFSET`
: This is a type−safe representation of an offset into a ***MySQL*** field list. (Used by `mysql_field_seek()`.) Offsets are field numbers within a row, beginning at zero.

`my_ulonglong`
: The type used for the number of rows and for `mysql_affected_rows()`, `mysql_num_rows()` and `mysql_insert_id()`. This type provides a range of `0` to `1.84e19`. On some systems, attempting to print a value of type `my_ulonglong` will not work. To print

such a value, convert it to `unsigned long` and use a `%lu` print format. Example:

```
printf (Number of rows: %lu\n", (unsigned long) mysql_num_rows(result));
```

The `MYSQL_FIELD` structure contains the members listed below:

`char * name`
The name of the field.

`char * table`
The name of the table containing this field, if it isn't a calculated field. For calculated fields, the `table` value is a `NULL` pointer.

`char * def`
The default value of this field (set only if you use `mysql_list_fields()`).

`enum enum_field_types type`
The type of the field. The `type` value may be one of the following:

Type value	***Type meaning***
`FIELD_TYPE_TINY`	`TINYINT` field
`FIELD_TYPE_SHORT`	`SMALLINT` field
`FIELD_TYPE_LONG`	`INTEGER` field
`FIELD_TYPE_INT24`	`MEDIUMINT` field
`FIELD_TYPE_LONGLONG`	`BIGINT` field
`FIELD_TYPE_DECIMAL`	`DECIMAL` or `NUMERIC` field
`FIELD_TYPE_FLOAT`	`FLOAT` field
`FIELD_TYPE_DOUBLE`	`DOUBLE` or `REAL` field
`FIELD_TYPE_TIMESTAMP`	`TIMESTAMP` field
`FIELD_TYPE_DATE`	`DATE` field
`FIELD_TYPE_TIME`	`TIME` field
`FIELD_TYPE_DATETIME`	`DATETIME` field
`FIELD_TYPE_YEAR`	`YEAR` field
`FIELD_TYPE_STRING`	String (`CHAR` or `VARCHAR`) field
`FIELD_TYPE_BLOB`	`BLOB` or `TEXT` field (use `max_length` to determine the maximum length)
`FIELD_TYPE_SET`	`SET` field
`FIELD_TYPE_ENUM`	`ENUM` field
`FIELD_TYPE_NULL`	`NULL`−type field
`FIELD_TYPE_CHAR`	Deprecated; use `FIELD_TYPE_TINY` instead

You can use the `IS_NUM()` macro to test whether or not a field has a numeric type. Pass the `type` value to `IS_NUM()` and it will evaluate to TRUE if the field is numeric:

```
if (IS_NUM(field−62;type))
    printf("Field is numeric\n");
```

`unsigned int length`
The width of the field.

`unsigned int max_length`
The maximum width of the field for the result set. If you used `mysql_list_fields()`, this contains the maximum length for the field.

`unsigned int flags`
Different bit−flags for the field. The `flags` value may have zero or more of the following bits set:

Flag value	***Flag meaning***
`NOT_NULL_FLAG`	Field can't be `NULL`
`PRI_KEY_FLAG`	Field is part of a primary key
`UNIQUE_KEY_FLAG`	Field is part of a unique key
`MULTIPLE_KEY_FLAG`	Field is part of a non−unique key.
`UNSIGNED_FLAG`	Field has the `UNSIGNED` attribute
`ZEROFILL_FLAG`	Field has the `ZEROFILL` attribute

BINARY_FLAG	Field has the BINARY attribute
AUTO_INCREMENT_FLAG	Field has the AUTO_INCREMENT attribute
ENUM_FLAG	Field is an ENUM (deprecated)
BLOB_FLAG	Field is a BLOB or TEXT (deprecated)
TIMESTAMP_FLAG	Field is a TIMESTAMP (deprecated)

Use of the BLOB_FLAG, ENUM_FLAG and TIMESTAMP_FLAG flags is deprecated because they indicate the type of a field rather than an attribute of its type. It is preferable to test field->type against FIELD_TYPE_BLOB, FIELD_TYPE_ENUM or FIELD_TYPE_TIMESTAMP instead. The example below illustrates a typical use of the flags value:

```
if (field−62;flags 38; NOT_NULL_FLAG)
    printf("Field can't be null\n");
```

You may use the following convenience macros to determine the boolean status of the flags value:

IS_NOT_NULL(flags)	True if this field is defined as NOT NULL
IS_PRI_KEY(flags)	True if this field is a primary key
IS_BLOB(flags)	True if this field is a BLOB or TEXT (deprecated; test field->type instead)

unsigned int decimals
: The number of decimals for numeric fields.

20.3 présentation des fonctions C API

The functions available in the C API are listed below and are described in greater detail in the next section. 20.4 Descriptions des fonctions C API.

mysql_affected_rows()	Returns the number of rows affected by the last UPDATE, DELETE or INSERT query.
mysql_close()	Closes a server connection.
mysql_connect()	Connects to a ***MySQL*** server. This function is deprecated; use mysql_real_connect() instead.
mysql_change_user()	Change user and database on an open connection.
mysql_create_db()	Creates a database. This function is deprecated; use the SQL command CREATE DATABASE instead.
mysql_data_seek()	Seeks to an arbitrary row in a query result set.
mysql_debug()	Does a DBUG_PUSH with the given string.
mysql_drop_db()	Drops a database. This function is deprecated; use the SQL command DROP DATABASE instead.
mysql_dump_debug_info()	Makes the server write debug information to the log.
mysql_eof()	Determines whether or not the last row of a result set has been read. This function is deprecated; mysql_errno() or mysql_error() may be used instead.
mysql_errno()	Returns the error number for the most recently invoked ***MySQL*** function.
mysql_error()	Returns the error message for the most recently invoked ***MySQL*** function.
mysql_escape_string()	Escapes special characters in a string for use in a SQL statement.
mysql_fetch_field()	Returns the type of the next table field.
mysql_fetch_field_direct()	Returns the type of a table field, given a field number.
mysql_fetch_fields()	Returns an array of all field structures.
mysql_fetch_lengths()	Returns the lengths of all columns in the current row.
mysql_fetch_row()	Fetches the next row from the result set.
mysql_field_seek()	Puts the column cursor on a specified column.
mysql_field_count()	Returns the number of result columns for the most recent query.
mysql_field_tell()	Returns the position of the field cursor used for the last mysql_fetch_field().

mysql_free_result()	Frees memory used by a result set.
mysql_get_client_info()	Returns client version information.
mysql_get_host_info()	Returns a string describing the connection.
mysql_get_proto_info()	Returns the protocol version used by the connection.
mysql_get_server_info()	Returns the server version number.
mysql_info()	Returns information about the most recently executed query.
mysql_init()	Gets or initializes a `MYSQL` structure.
mysql_insert_id()	Returns the ID generated for an `AUTO_INCREMENT` column by the previous query.
mysql_kill()	Kill a given thread.
mysql_list_dbs()	Returns database names matching a simple regular expression.
mysql_list_fields()	Returns field names matching a simple regular expression.
mysql_list_processes()	Returns a list of the current server threads.
mysql_list_tables()	Returns table names matching a simple regular expression.
mysql_num_fields()	Returns the number of columns in a result set.
mysql_num_rows()	Returns the number of rows in a result set.
mysql_options()	Set connect options for `mysql_connect()`.
mysql_ping()	Checks whether or not the connection to the server is working, reconnecting as necessary.
mysql_query()	Executes a SQL query specified as a null−terminated string.
mysql_real_connect()	Connects to a ***MySQL*** server.
mysql_real_query()	Executes a SQL query specified as a counted string.
mysql_reload()	Tells the server to reload the grant tables.
mysql_row_seek()	Seeks to a row in a result set, using value returned from `mysql_row_tell()`.
mysql_row_tell()	Returns the row cursor position.
mysql_select_db()	Connects to a database.
mysql_shutdown()	Shuts down the database server.
mysql_stat()	Returns the server status as a string.
mysql_store_result()	Retrieves a complete result set to the client.
mysql_thread_id()	Returns the current thread ID.
mysql_use_result()	Initiates a row−by−row result set retrieval.

To connect to the server, call `mysql_init()` to initialize a connection handler, then call `mysql_real_connect()` with that handler (along with other information such as the hostname, user name and password). When you are done with the connection, call `mysql_close()` to terminate it.

While a connection is active, the client may send SQL requêtesto the server using `mysql_query()` or `mysql_real_query()`. The difference between the two is that `mysql_query()` expects the query to be specified as a null−terminated string whereas `mysql_real_query()` expects a counted string. If the string contains binary data (which may include null bytes), you must use `mysql_real_query()`.

For each non−`SELECT` query (e.g., `INSERT`, `UPDATE`, `DELETE`), you can found out how many rows were affected (changed) by calling `mysql_affected_rows()`.

For `SELECT` queries, you retrieve the selected rows as a result set. (Note that some statements are `SELECT`−like in that they return rows. These include `SHOW`, `DESCRIBE` and `EXPLAIN`. They should be treated the same way as `SELECT` statements.)

There are two ways for a client to process result sets. One way is to retrieve the entire result set all at once by calling `mysql_store_result()`. This function acquires from the server all the rows returned by the query and stores them in the client. The second way is for the client to initiate a row−by−row result set retrieval by calling `mysql_use_result()`. This function initializes the retrieval, but does not actually get any rows from the server.

In both cases, you access rows by calling `mysql_fetch_row()`. With `mysql_store_result()`, `mysql_fetch_row()` accesses rows that have already been fetched from the server. With `mysql_use_result()`, `mysql_fetch_row()` actually retrieves the row from the server. Information about as the size of the data values in each row is available by calling `mysql_fetch_lengths()`.

After you are done with a result set, call `mysql_free_result()` to free the memory used for it.

The two retrieval mechanisms are complementary. Client programs should choose the approach that is most appropriate for their requirements. In practice, clients tend to use `mysql_store_result()` more commonly.

An advantage of `mysql_store_result()` is that since the rows have all been fetched to the client, you not only can access rows sequentially, you can move back and forth in the result set using `mysql_data_seek()` or `mysql_row_seek()` to change the current row position within the result set. You can also find out how many rows there are by calling `mysql_num_rows()`. On the other hand, the memory requirements for `mysql_store_result()` may be very high for large result sets and you are more likely to encounter out−of−memory conditions.

An advantage of `mysql_use_result()` is that the client requires less memory for the result set since it maintains only one row at a time (and since there is less allocation overhead, `mysql_use_result()` can be faster). Disadvantages are that you must process each row quickly to avoid tying up the server, you don't have random access to rows within the result set (you can only access rows sequentially), and you don't know how many rows are in the result set until you have retrieved them all. Furthermore, you *must* retrieve all the rows even if you determine in mid−retrieval that you've found the information you were looking for.

The API makes it possible for clients to respond appropriately to requêtes(retrieving rows only as necessary) without knowing whether or not the query is a `SELECT`. You can do this by calling `mysql_store_result()` after each `mysql_query()` (or `mysql_real_query()`). If the result set call succeeds, the query was a `SELECT` and you can read the rows. If the result set call fails, call `mysql_field_count()` to determine whether or not a result was actually to be expected. If `mysql_field_count()` returns zero, the query returned no data (indicating that it was an `INSERT`, `UPDATE`, `DELETE`, etc.), and thus not expected to return rows. If `mysql_field_count()` is non−zero, the query should have returned rows, but didn't. This indicates that the query was a `SELECT` that failed. See the description for `mysql_field_count()` for an example of how this can be done.

Both `mysql_store_result()` and `mysql_use_result()` allow you to obtain information about the fields that make up the result set (the number of fields, their names and types, etc.). You can access field information sequentially within the row by calling `mysql_fetch_field()` repeatedly, or by field number within the row by calling `mysql_fetch_field_direct()`. The current field cursor position may be changed by calling `mysql_field_seek()`. Setting the field cursor affects subsequent calls to `mysql_fetch_field()`. You can also get information for fields all at once by calling `mysql_fetch_fields()`.

For detecting and reporting errors, ***MySQL*** provides access to error information by means of the `mysql_errno()` and `mysql_error()` functions. These return the error code or error message for the

most recently invoked function that can succeed or fail, allowing you to determine when an error occurred and what it was.

20.4 Descriptions des fonctions C API

In the descriptions below, a parameter or return value of `NULL` means `NULL` in the sense of the C programming language, not a ***MySQL*** `NULL` value.

Functions that return a value generally return a pointer or an integer. Unless specified otherwise, functions returning a pointer return a non−`NULL` value to indicate success or a `NULL` value to indicate an error, and functions returning an integer return zero to indicate success or non−zero to indicate an error. Note that ``non−zero'' means just that. Unless the function description says otherwise, do not test against a value other than zero:

```
if (result)                  /* correct */
    ... error ...

if (result 60; 0)              /* incorrect */
    ... error ...

if (result == −1)            /* incorrect */
    ... error ...
```

When a function returns an error, the ***Errors*** subsection of the function description lists the possible types of errors. You can find out which of these occurred by calling `mysql_errno()`. A string representation of the error may be obtained by calling `mysql_error()`.

20.4.1 `mysql_affected_rows()`

```
my_ulonglong mysql_affected_rows(MYSQL *mysql)
```

20.4.1.1 Description

Returns the number of rows affected (changed) by the last `UPDATE`, `DELETE` or `INSERT` query. May be called immediately after `mysql_query()` for `UPDATE`, `DELETE` or `INSERT` statements. For `SELECT` statements, `mysql_affected_rows()` works like `mysql_num_rows()`.

`mysql_affected_rows()` is currently implemented as a macro.

20.4.1.2 Return values

An integer greater than zero indicates the number of rows affected or retrieved. Zero indicates that no records matched the `WHERE` clause in the query or that no query has yet been executed. −1 indicates that the query returned an error or that, for a `SELECT` query, `mysql_affected_rows()` was called prior to calling `mysql_store_result()`.

20.4.1.3 Errors

None.

20.4.1.4 Example

```
mysql_query("UPDATE products SET cost=cost*1.25 WHERE group=10");
printf("%d products updated",mysql_affected_rows();
```

20.4.2 mysql_close()

```
void mysql_close(MYSQL *mysql)
```

20.4.2.1 Description

Closes a previously opened connection. `mysql_close()` also deallocates the connection handle pointed to by `mysql` if the handle was allocated automatically by `mysql_init()` or `mysql_real_connect()`.

20.4.2.2 Return values

None.

20.4.2.3 Errors

`CR_COMMANDS_OUT_OF_SYNC`
: Commands were executed in an improper order.

`CR_SERVER_GONE_ERROR`
: The ***MySQL*** server has gone away.

`CR_SERVER_LOST`
: The connection to the server was lost during the query.

`CR_UNKNOWN_ERROR`
: An unknown error occurred.

20.4.3 mysql_connect()

```
MYSQL *mysql_connect(MYSQL *mysql, const char *host, const char *user,
const char *passwd)
```

20.4.3.1 Description

This function is deprecated. It is preferable to use `mysql_real_connect()` instead.

`mysql_connect()` attempts to establish a connection to a ***MySQL*** database engine running on `host`. `mysql_connect()` must complete successfully before you can execute any of the other API functions, with the exception of `mysql_get_client_info()`.

The meanings of the parameters are the same as for the corresponding parameters for `mysql_real_connect()`. See the description of that function for more information.

20.4.3.2 Return values

Same as for `mysql_real_connect()`.

20.4.3.3 Errors

Same as for `mysql_real_connect()`.

20.4.4 `mysql_change_user()`

```
my_bool mysql_change_user(MYSQL *mysql, const char *user, const char
*password, const char *db)
```

20.4.4.1 Description

Changes the user and causes the database specified by `db` to become the default (current) database on the connection specified by `mysql`. In subsequent queries, this database is the default for table references that do not include an explicit database specifier.

This function was introduced in ***MySQL*** 3.23.3

`mysql_cohange_user()` fails unless the connected user can be authenticated or if he doesn't have permission to use the database. In this case the user and database is not changed

20.4.4.2 Return values

Zero for success. Non−zero if an error occurred.

20.4.4.3 Errors

The same that you can get from `mysql_real_connect()`

CR_COMMANDS_OUT_OF_SYNC
: Commands were executed in an improper order.

CR_SERVER_GONE_ERROR
: The ***MySQL*** server has gone away.

CR_SERVER_LOST
: The connection to the server was lost during the query.

CR_UNKNOWN_ERROR
: An unknown error occurred.

ER_UNKNOWN_COM_ERROR
: The ***MySQL*** server doesn't implement this command (probably an old server)

ER_ACCESS_DENIED_ERROR
: The user or password was wrong.

ER_BAD_DB_ERROR
: The database didn't exists.

ER_DBACCESS_DENIED_ERROR
: The user did not have access rights to the database.

ER_WRONG_NOM_BASE_DE_DONNEES
: The database name was too long.

20.4.4.4 Example

```
if (mysql_change_user(38;mysql, "user", "password", "new_database"))
{
   fprintf(stderr, "Failed to change user.  Error: %s\n",
           mysql_error();
}
```

20.4.5 mysql_create_db()

```
int mysql_create_db(MYSQL *mysql, const char *db)
```

20.4.5.1 Description

Creates the database named by the `db` parameter.

This function is deprecated. It is preferable to use `mysql_query()` to issue a SQL `CREATE DATABASE` statement instead.

20.4.5.2 Return values

Zero if the database was created successfully. Non−zero if an error occurred.

20.4.5.3 Errors

`CR_COMMANDS_OUT_OF_SYNC`
: Commands were executed in an improper order.

`CR_SERVER_GONE_ERROR`
: The ***MySQL*** server has gone away.

`CR_SERVER_LOST`
: The connection to the server was lost during the query.

`CR_UNKNOWN_ERROR`
: An unknown error occurred.

20.4.5.4 Example

```
if(mysql_create_db(38;mysql, "my_database"))
{
   fprintf(stderr, "Failed to create new database.  Error: %s\n",
           mysql_error();
}
```

20.4.6 mysql_data_seek()

```
void mysql_data_seek(MYSQL_RES *result, unsigned int offset)
```

20.4.6.1 Description

Seeks to an arbitrary row in a query result set. This requires that the result set structure contains the entire result of the query, so `mysql_data_seek()` may be used in conjunction only with `mysql_store_result()`, not with `mysql_use_result()`.

The offset should be a value in the range from 0 to `mysql_num_rows(result)−1`.

20.4.6.2 Return values

None.

20.4.6.3 Errors

None.

20.4.7 mysql_debug()

```
void mysql_debug(char *debug)
```

20.4.7.1 Description

Does a DBUG_PUSH with the given string. mysql_debug() uses the Fred Fish debug library. To use this function, you must compile the client library to support debugging. G.1 Debugguer un serveur MySQL. G.2 Debugguer un client MySQL.

20.4.7.2 Return values

None.

20.4.7.3 Errors

None.

20.4.7.4 Example

The call shown below causes the client library to generate a trace file in `/tmp/client.trace' on the client machine:

```
mysql_debug("d:t:O,/tmp/client.trace");
```

20.4.8 mysql_drop_db()

```
int mysql_drop_db(MYSQL *mysql, const char *db)
```

20.4.8.1 Description

Drops the database named by the db parameter.

This function is deprecated. It is preferable to use mysql_query() to issue a SQL DROP DATABASE statement instead.

20.4.8.2 Return values

Zero if the database was dropped successfully. Non−zero if an error occurred.

20.4.8.3 Errors

CR_COMMANDS_OUT_OF_SYNC
: Commands were executed in an improper order.

CR_SERVER_GONE_ERROR
: The ***MySQL*** server has gone away.

CR_SERVER_LOST
: The connection to the server was lost during the query.

CR_UNKNOWN_ERROR
: An unknown error occurred.

20.4.8.4 Example

```
if(mysql_drop_db(38;mysql, "my_database"))
  fprintf(stderr, "Failed to drop the database: Error: %s\n",
          mysql_error();
```

20.4.9 mysql_dump_debug_info()

```
int mysql_dump_debug_info(MYSQL *mysql)
```

20.4.9.1 Description

Instructs the server to write some debug information to the log. The connected user must have the ***process*** privilege for this to work.

20.4.9.2 Return values

Zero if the command was successful. Non−zero if an error occurred.

20.4.9.3 Errors

CR_COMMANDS_OUT_OF_SYNC
: Commands were executed in an improper order.

CR_SERVER_GONE_ERROR
: The ***MySQL*** server has gone away.

CR_SERVER_LOST
: The connection to the server was lost during the query.

CR_UNKNOWN_ERROR
: An unknown error occurred.

20.4.10 mysql_eof()

```
my_bool mysql_eof(MYSQL_RES *result)
```

20.4.10.1 Description

This function is deprecated. `mysql_errno()` or `mysql_error()` may be used instead.

`mysql_eof()` determines whether or not the last row of a result set has been read.

If you acquire a result set from a successful call to `mysql_store_result()`, the client receives the entire set in one operation. In this case, a `NULL` return from `mysql_fetch_row()` always means the end of the result set has been reached and it is unnecessary to call `mysql_eof()`.

On the other hand, if you use `mysql_use_result()` to initiate a result set retrieval, the rows of the set are obtained from the server one by one as you call `mysql_fetch_row()` repeatedly. Because an error may occur on the connection during this process, a `NULL` return value from `mysql_fetch_row()` does

not necessarily mean the end of the result set was reached normally. In this case, you can use `mysql_eof()` to determine what happened. `mysql_eof()` returns a non−zero value if the end of the result set was reached and zero if an error occurred.

Historically, `mysql_eof()` predates the standard ***MySQL*** error functions `mysql_errno()` and `mysql_error()`. Since those error functions provide the same information, their use is preferred over `mysql_eof()`, which is now deprecated. (In fact, they provide more information, since `mysql_eof()` returns only a boolean value whereas the error functions indicate a reason for the error when one occurs.)

20.4.10.2 Return values

Zero if an error occurred. Non−zero if the end of the result set has been reached.

20.4.10.3 Errors

None.

20.4.10.4 Example

The following example shows how you might use `mysql_eof()`:

```
mysql_query("SELECT * FROM some_table");
result = mysql_use_result(;
while((row = mysql_fetch_row(result)))
{
    // do something with data
}
if(!mysql_eof(result))  // mysql_fetch_row() failed due to an error
{
    fprintf(stderr, "Error: %s\n", mysql_error();
}
```

However, you can achieve the same effect with the standard ***MySQL*** error functions:

```
mysql_query("SELECT * FROM some_table");
result = mysql_use_result(;
while((row = mysql_fetch_row(result)))
{
    // do something with data
}
if(mysql_errno(38;mysql))  // mysql_fetch_row() failed due to an error
{
    fprintf(stderr, "Error: %s\n", mysql_error();
}
```

20.4.11 `mysql_errno()`

```
unsigned int mysql_errno(MYSQL *mysql)
```

20.4.11.1 Description

For the connection specified by `mysql`, `mysql_errno()` returns the error code for the most recently invoked API function that can succeed or fail. A return value of zero means that no error occurred. Client error message numbers are listed in the ***MySQL*** `` `errmsg.h' `` header file. Server error message numbers are listed in `` `mysqld_error.h' ``

20.4.11.2 Return values:

An error code value. Zero if no error occurred.

20.4.11.3 Errors

None.

20.4.12 `mysql_error()`

```
char *mysql_error(MYSQL *mysql)
```

20.4.12.1 Description

For the connection specified by `mysql`, `mysql_error()` returns the error message for the most recently invoked API function that can succeed or fail. An empty string (`""`) is returned if no error occurred. This means the following two tests are equivalent:

```
if(mysql_errno(38;mysql))
{
    // an error occurred
}

if(mysql_error(38;mysql)[0] != '\0')
{
    // an error occurred
}
```

The language of the client error messages may be changed by recompiling the ***MySQL*** client library. Currently you can choose error messages in several different languages. 9.1 Quels sont les langues supportés par MySQL?.

20.4.12.2 Return values

A character string that describes the error. An empty string if no error occurred.

20.4.12.3 Errors

None.

20.4.13 mysql_escape_string()

```
unsigned int mysql_escape_string(char *to, const char *from, unsigned
int length)
```

20.4.13.1 Description

Encodes the string in `from` to an escaped SQL string that can be sent to the server in a SQL statement, and places the result in `to`. Characters encoded are `NUL` (ASCII 0), `` `\n' ``, `` `\r' ``, `` `\' `` and `` `'' `` (7.1 Syntaxe des chaînes et nombres).

The string pointed to by `from` must be `length` bytes long (not including the terminating null byte). You must allocate the `to` buffer to be at least `length*2+1` bytes long. When `mysql_escape_string()` returns, the contents of `to` will be a null−terminated string. The return value is the length of the encoded string, not including the terminating null character.

20.4.13.2 Example

```
char query[1000],*end;

end = strmov(query,"INSERT INTO test_table values(");
*end++ = '\'';
end += mysql_escape_string(end,"What's this",11);
*end++ = '\'';
*end++ = ',';
*end++ = '\'';
end += mysql_escape_string(end,"binary data: \0\r\n",16);
*end++ = '\'';
*end++ = ')';

if (mysql_real_query(38;mysql,query,(unsigned int) (end − query)))
{
   fprintf(stderr, "Failed to insert row, Error: %s\n",
           mysql_error();
}
```

The `strmov()` function used in the example is included in the `mysqlclient` library and works like `strcpy()` but returns a pointer to the terminating null of the first parameter.

20.4.13.3 Return values

The length of the value placed into `to`, not including the terminating null character.

20.4.13.4 Errors

None.

20.4.14 mysql_fetch_field()

```
MYSQL_FIELD *mysql_fetch_field(MYSQL_RES *result)
```

20.4.14.1 Description

Returns the definition of one column of a result set as a `MYSQL_FIELD` structure. Call this function repeatedly to retrieve information about all columns in the result set. `mysql_fetch_field()` returns `NULL` when no more fields are left.

`mysql_fetch_field()` is reset to return information about the first field each time you execute a new `SELECT` query. The field returned by `mysql_fetch_field()` is also affected by calls to `mysql_field_seek()`.

If you've called `mysql_query()` to perform a `SELECT` on a table but have not called `mysql_store_result()`, ***MySQL*** returns the default blob length (8K bytes) if you call `mysql_fetch_field()` to ask for the length of a `BLOB` field. (The 8K size is chosen because ***MySQL*** doesn't know the maximum length for the `BLOB`. This should be made configurable sometime.) Once you've retrieved the result set, `field->max_length` contains the length of the largest value for this column in the specific query.

20.4.14.2 Return values

The `MYSQL_FIELD` structure for the current column. `NULL` if no columns are left.

20.4.14.3 Errors

None.

20.4.14.4 Example

```
MYSQL_FIELD *field;

while((field = mysql_fetch_field(result)))
{
    printf("field name %s\n", field-62;name);
}
```

20.4.15 `mysql_fetch_fields()`

```
MYSQL_FIELD *mysql_fetch_fields(MYSQL_RES *result)
```

20.4.15.1 Description

Returns an array of all `MYSQL_FIELD` structures for a result set. Each structure provides the field definition for one column of the result set.

20.4.15.2 Return values

An array of `MYSQL_FIELD` structures for all columns of a result set.

20.4.15.3 Errors

None.

20.4.15.4 Example

```
unsigned int num_fields;
unsigned int i;
MYSQL_FIELD *fields;

num_fields = mysql_num_fields(result);
fields = mysql_fetch_fields(result);
for(i = 0; i 60; num_fields; i++)
{
   printf("Field %u is %s\n", i, fields[i].name);
}
```

20.4.16 mysql_fetch_field_direct()

```
MYSQL_FIELD *mysql_fetch_field_direct(MYSQL_RES *result, unsigned int
fieldnr)
```

20.4.16.1 Description

Given a field number `fieldnr` for a column within a result set, returns that column's field definition as a `MYSQL_FIELD` structure. You may use this function to retrieve the definition for an arbitrary column. The value of `fieldnr` should be in the range from 0 to `mysql_num_fields(result)−1`.

20.4.16.2 Return values

The `MYSQL_FIELD` structure for the specified column.

20.4.16.3 Errors

None.

20.4.16.4 Example

```
unsigned int num_fields;
unsigned int i;
MYSQL_FIELD *field;

num_fields = mysql_num_fields(result);
for(i = 0; i 60; num_fields; i++)
{
    field = mysql_fetch_field_direct(result, i);
    printf("Field %u is %s\n", i, field−62;name);
}
```

20.4.17 mysql_fetch_lengths()

```
unsigned long *mysql_fetch_lengths(MYSQL_RES *result)
```

20.4.17.1 Description

Returns the lengths of the columns of the current row within a result set. If you plan to copy field values, this length information is also useful for optimization, because you can avoid calling `strlen()`. In addition, if the result set contains binary data, you *must* use this function to determine the size of the data, because `strlen()` returns incorrect results for any field containing null characters.

The length for empty columns and for columns containing `NULL` values is zero. To see how to distinguish these two cases, see the description for `mysql_fetch_row()`.

20.4.17.2 Return values

An array of unsigned long integers representing the size of each column (not including any terminating null characters). `NULL` if an error occurred.

20.4.17.3 Errors

`mysql_fetch_lengths()` is valid only for the current row of the result set. It returns `NULL` if you call it before calling `mysql_fetch_row()` or after retrieving all rows in the result.

20.4.17.4 Example

```
MYSQL_ROW row;
unsigned long *lengths;
unsigned int num_fields;
unsigned int i;

row = mysql_fetch_row(result);
if (row)
{
    num_fields = mysql_num_fields(result);
    lengths = mysql_fetch_lengths(result);
    for(i = 0; i 60; num_fields; i++)
    {
         printf("Column %u is %lu bytes in length.\n", i, lengths[i]);
    }
}
```

20.4.18 `mysql_fetch_row()`

```
MYSQL_ROW mysql_fetch_row(MYSQL_RES *result)
```

20.4.18.1 Description

Retrieves the next row of a result set. When used after `mysql_store_result()`, `mysql_fetch_row()` returns `NULL` when there are no more rows to retrieve. When used after `mysql_use_result()`, `mysql_fetch_row()` returns `NULL` when there are no more rows to retrieve or if an error occurred.

The number of values in the row is given by `mysql_num_fields(result)`. If `row` holds the return value from a call to `mysql_fetch_row()`, pointers to the values are accessed as `row[0]` to `row[mysql_num_fields(result)−1]`. `NULL` values in the row are indicated by `NULL` pointers.

The lengths of the field values in the row may be obtained by calling `mysql_fetch_lengths()`. Empty fields and fields containing `NULL` both have length 0; you can distinguish these by checking the pointer for the field value. If the pointer is `NULL`, the field is `NULL`; otherwise the field is empty.

20.4.18.2 Return values

A `MYSQL_ROW` structure for the next row. `NULL` if there are no more rows to retrieve or if an error occurred.

20.4.18.3 Errors

`CR_SERVER_LOST`
: The connection to the server was lost during the query.

`CR_UNKNOWN_ERROR`
: An unknown error occurred.

20.4.18.4 Example

```
MYSQL_ROW row;
unsigned int num_fields;
unsigned int i;

num_fields = mysql_num_fields(result);
while ((row = mysql_fetch_row(result)))
{
   unsigned long *lengths;
   lengths = mysql_fetch_lengths(result);
   for(i = 0; i 60; num_fields; i++)
   {
       printf("[%.*s] ", (int) lengths[i], row[i] ? row[i] : "NULL");
   }
   printf("\n");
}
```

20.4.19 `mysql_field_count()`

```
unsigned int mysql_field_count(MYSQL *mysql)
```

If you are using a version of ***MySQL*** earlier than 3.22.24, you should use `unsigned int mysql_num_fields(MYSQL *mysql)` instead.

20.4.19.1 Description

Returns the number of columns for the most recent query on the connection.

The normal use of this function is when `mysql_store_result()` returned `NULL` (and thus you have no result set pointer). In this case, you can call `mysql_field_count()` to determine whether or not `mysql_store_result()` should have produced a non−empty result. This allows the client program to take proper action without knowing whether or not the query was a `SELECT` (or `SELECT`−like) statement. The example shown below illustrates how this may be done.

`NULL mysql_store_result()`.

20.4.19.2 Return values

An unsigned integer representing the number of fields in a result set.

20.4.19.3 Errors

None.

20.4.19.4 Example

```
MYSQL_RES *result;
unsigned int num_fields;
unsigned int num_rows;

if (mysql_query(38;mysql,query_string))
{
    // error
}
else // query succeeded, process any data returned by it
{
    result = mysql_store_result(;
    if (result)  // there are rows
    {
        num_fields = mysql_num_fields(result);
        // retrieve rows, then call mysql_free_result(result)
    }
    else  // mysql_store_result() returned nothing; should it have?
    {
        if(mysql_field_count(38;mysql) == 0)
        {
            // query does not return data
            // (it was not a SELECT)
            num_rows = mysql_affected_rows(;
        }
        else // mysql_store_result() should have returned data
        {
            fprintf(stderr, "Error: %s\n", mysql_error();
        }
    }
}
```

An alternative is to replace the `mysql_field_count(&mysql)` call with `mysql_errno(&mysql)`. In this case, you are checking directly for an error from `mysql_store_result()` rather than inferring from the value of `mysql_field_count()` whether or not the statement was a `SELECT`.

20.4.20 `mysql_field_seek()`

```
MYSQL_FIELD_OFFSET mysql_field_seek(MYSQL_RES *result,
MYSQL_FIELD_OFFSET offset)
```

20.4.20.1 Description

Sets the field cursor to the given offset. The next call to `mysql_fetch_field()` will retrieve the field definition of the column associated with that offset.

To seek to the beginning of a row, pass an `offset` value of zero.

20.4.20.2 Return values

The previous value of the field cursor.

20.4.20.3 Errors

None.

20.4.21 `mysql_field_tell()`

```
MYSQL_FIELD_OFFSET mysql_field_tell(MYSQL_RES *result)
```

20.4.21.1 Description

Returns the position of the field cursor used for the last `mysql_fetch_field()`. This value can be used as an argument to `mysql_field_seek()`.

20.4.21.2 Return values

The current offset of the field cursor.

20.4.21.3 Errors

None.

20.4.22 `mysql_free_result()`

```
void mysql_free_result(MYSQL_RES *result)
```

20.4.22.1 Description

Frees the memory allocated for a result set by `mysql_store_result()`, `mysql_use_result()`, `mysql_list_dbs()`, etc. When you are done with a result set, you must free the memory it uses by calling `mysql_free_result()`.

20.4.22.2 Return values

None.

20.4.22.3 Errors

None.

20.4.23 mysql_get_client_info()

```
char *mysql_get_client_info(void)
```

20.4.23.1 Description

Returns a string that represents the client library version.

20.4.23.2 Return values

A character string that represents the ***MySQL*** client library version.

20.4.23.3 Errors

None.

20.4.24 mysql_get_host_info()

```
char *mysql_get_host_info(MYSQL *mysql)
```

20.4.24.1 Description

Returns a string describing the type of connection in use, including the server host name.

20.4.24.2 Return values

A character string representing the server host name and the connection type.

20.4.24.3 Errors

None.

20.4.25 mysql_get_proto_info()

```
unsigned int mysql_get_proto_info(MYSQL *mysql)
```

20.4.25.1 Description

Returns the protocol version used by current connection.

20.4.25.2 Return values

An unsigned integer representing the protocol version used by the current connection.

20.4.25.3 Errors

None.

20.4.26 mysql_get_server_info()

```
char *mysql_get_server_info(MYSQL *mysql)
```

20.4.26.1 Description

Returns a string that represents the server version number.

20.4.26.2 Return values

A character string that represents the server version number.

20.4.26.3 Errors

None.

20.4.27 mysql_info()

```
char *mysql_info(MYSQL *mysql)
```

20.4.27.1 Description

Retrieves a string providing information about the most recently executed query, but only for the statements listed below. For other statements, `mysql_info()` returns `NULL`. The format of the string varies depending on the type of query, as described below. The numbers are illustrative only; the string will contain values appropriate for the query.

`INSERT INTO ... SELECT ...`
String format: `Records: 100 Duplicates: 0 Warnings: 0`
`INSERT INTO ... VALUES (...),(...),(...)...`
String format: `Records: 3 Duplicates: 0 Warnings: 0`
`LOAD DATA INFILE ...`
String format: `Records: 1 Deleted: 0 Skipped: 0 Warnings: 0`
`ALTER TABLE`
String format: `Records: 3 Duplicates: 0 Warnings: 0`

Note that `mysql_info()` returns a non−`NULL` value for the `INSERT ... VALUES` statement only if multiple value lists are specified in the statement.

20.4.27.2 Return values

A character string representing additional information about the most recently executed query. `NULL` if no information is available for the query.

20.4.27.3 Errors

None.

20.4.28 mysql_init()

```
MYSQL *mysql_init(MYSQL *mysql)
```

20.4.28.1 Description

Allocates or initializes a MYSQL object suitable for mysql_real_connect(). If mysql is a NULL pointer, the function allocates, initializes and returns a new object. Otherwise the object is initialized and the address of the object is returned. If mysql_init() allocates a new object, it will be freed when mysql_close() is called to close the connection.

20.4.28.2 Return values

An initialized MYSQL* handle. NULL if there was insufficient memory to allocate a new object.

20.4.28.3 Errors

In case of insufficient memory, NULL is returned.

20.4.29 mysql_insert_id()

```
my_ulonglong mysql_insert_id(MYSQL *mysql)
```

20.4.29.1 Description

Returns the ID generated for an AUTO_INCREMENT column by the previous query. Use this function after you have performed an INSERT query into a table that contains an AUTO_INCREMENT field.

Note that mysql_insert_id() returns 0 if the previous query does not generate an AUTO_INCREMENT value. If you need to save the value for later, be sure to call mysql_insert_id() immediately after the query that generates the value.

Also note that the value of the SQL LAST_INSERT_ID() function always contains the most recently generated AUTO_INCREMENT value, and is not reset between requêtessince the value of that function is maintained in the server.

20.4.29.2 Return values

The value of the AUTO_INCREMENT field that was updated by the previous query. Returns zero if there was no previous query on the connection or if the query did not update an AUTO_INCREMENT value.

20.4.29.3 Errors

None.

20.4.30 mysql_kill()

```
int mysql_kill(MYSQL *mysql, unsigned long pid)
```

20.4.30.1 Description

Asks the server to kill the thread specified by `pid`.

20.4.30.2 Return values

Zero for success. Non−zero if an error occurred.

20.4.30.3 Errors

CR_COMMANDS_OUT_OF_SYNC
: Commands were executed in an improper order.

CR_SERVER_GONE_ERROR
: The ***MySQL*** server has gone away.

CR_SERVER_LOST
: The connection to the server was lost during the query.

CR_UNKNOWN_ERROR
: An unknown error occurred.

20.4.31 mysql_list_dbs()

```
MYSQL_RES *mysql_list_dbs(MYSQL *mysql, const char *wild)
```

20.4.31.1 Description

Returns a result set consisting of database names on the server that match the simple regular expression specified by the `wild` parameter. `wild` may contain the wildcard characters `` `%' `` or `` `_' ``, or may be a `NULL` pointer to match all databases. Calling `mysql_list_dbs()` is similar to executing the query `SHOW databases [LIKE wild]`.

You must free the result set with `mysql_free_result()`.

20.4.31.2 Return values

A `MYSQL_RES` result set for success. `NULL` if an error occurred.

20.4.31.3 Errors

CR_COMMANDS_OUT_OF_SYNC
: Commands were executed in an improper order.

CR_OUT_OF_MEMORY
: Out of memory.

CR_SERVER_GONE_ERROR
: The ***MySQL*** server has gone away.

CR_SERVER_LOST
: The connection to the server was lost during the query.

CR_UNKNOWN_ERROR
: An unknown error occurred.

20.4.32 mysql_list_fields()

```
MYSQL_RES *mysql_list_fields(MYSQL *mysql, const char *table, const char
*wild)
```

20.4.32.1 Description

Returns a result set consisting of field names in the given table that match the simple regular expression specified by the `wild` parameter. `wild` may contain the wildcard characters `%' or `_', or may be a `NULL` pointer to match all fields. Calling `mysql_list_fields()` is similar to executing the query `SHOW COLUMNS FROM nom_table [LIKE wild]`.

Note that it's recommended that you use `SHOW COLUMNS FROM nom_table` instead of `mysql_list_fields()`.

You must free the result set with `mysql_free_result()`.

20.4.32.2 Return values

A `MYSQL_RES` result set for success. `NULL` if an error occurred.

20.4.32.3 Errors

CR_COMMANDS_OUT_OF_SYNC
: Commands were executed in an improper order.

CR_SERVER_GONE_ERROR
: The ***MySQL*** server has gone away.

CR_SERVER_LOST
: The connection to the server was lost during the query.

CR_UNKNOWN_ERROR
: An unknown error occurred.

20.4.33 mysql_list_processes()

```
MYSQL_RES *mysql_list_processes(MYSQL *mysql)
```

20.4.33.1 Description

Returns a result set describing the current server threads. This is the same kind of information as that reported by `mysqladmin processlist`.

You must free the result set with `mysql_free_result()`.

20.4.33.2 Return values

A `MYSQL_RES` result set for success. `NULL` if an error occurred.

20.4.33.3 Errors

CR_COMMANDS_OUT_OF_SYNC
: Commands were executed in an improper order.

CR_SERVER_GONE_ERROR
: The ***MySQL*** server has gone away.

CR_SERVER_LOST
: The connection to the server was lost during the query.
CR_UNKNOWN_ERROR
: An unknown error occurred.

20.4.34 mysql_list_tables()

```
MYSQL_RES *mysql_list_tables(MYSQL *mysql, const char *wild)
```

20.4.34.1 Description

Returns a result set consisting of table names in the current database that match the simple regular expression specified by the `wild` parameter. `wild` may contain the wildcard characters `%' or `_', or may be a `NULL` pointer to match all tables. Calling `mysql_list_tables()` is similar to executing the query `SHOW tables [LIKE wild]`.

You must free the result set with `mysql_free_result()`.

20.4.34.2 Return values

A `MYSQL_RES` result set for success. `NULL` if an error occurred.

20.4.34.3 Errors

CR_COMMANDS_OUT_OF_SYNC
: Commands were executed in an improper order.
CR_SERVER_GONE_ERROR
: The ***MySQL*** server has gone away.
CR_SERVER_LOST
: The connection to the server was lost during the query.
CR_UNKNOWN_ERROR
: An unknown error occurred.

20.4.35 mysql_num_fields()

```
unsigned int mysql_num_fields(MYSQL_RES *result)
```

or

```
unsigned int mysql_num_fields(MYSQL *mysql)
```

The second form doesn't work on ***MySQL*** 3.22.24 or newer. To pass a `MYSQL*` argument, you must use `unsigned int mysql_field_count(MYSQL *mysql)` instead.

20.4.35.1 Description

Returns the number of columns in a result set.

Note that you can get the number of columns either from a pointer to a result set or to a connection handle. You would use the connection handle if `mysql_store_result()` returned `NULL` (and thus you have no result set pointer). In this case, you can call `mysql_field_count()` to determine whether or not `mysql_store_result()` should have produced a non−empty result. This allows the client program to

take proper action without knowing whether or not the query was a SELECT (or SELECT−like) statement. The example shown below illustrates how this may be done.

NULL mysql store result().

20.4.35.2 Return values

An unsigned integer representing the number of fields in a result set.

20.4.35.3 Errors

None.

20.4.35.4 Example

```
MYSQL_RES *result;
unsigned int num_fields;
unsigned int num_rows;

if (mysql_query(38;mysql,query_string))
{
    // error
}
else // query succeeded, process any data returned by it
{
    result = mysql_store_result(;
    if (result)  // there are rows
    {
        num_fields = mysql_num_fields(result);
        // retrieve rows, then call mysql_free_result(result)
    }
    else  // mysql_store_result() returned nothing; should it have?
    {
        if (mysql_errno(38;mysql))
        {
           fprintf(stderr, "Error: %s\n", mysql_error();
        }
        else if (mysql_field_count(38;mysql) == 0)
        {
            // query does not return data
            // (it was not a SELECT)
            num_rows = mysql_affected_rows(;
        }
    }
}
```

An alternative (if you KNOW that your query should have returned a result set) is to replace the `mysql_errno(&mysql)` call with a check if `mysql_field_count(&mysql)` is = 0. This will only happen if something went wrong.

20.4.36 mysql num rows()

```
my_ulonglong mysql_num_rows(MYSQL_RES *result)
```

20.4.36.1 Description

Returns the number of rows in the result set.

The use of `mysql_num_rows()` depends on whether you use `mysql_store_result()` or `mysql_use_result()` to return the result set. If you use `mysql_store_result()`, `mysql_num_rows()` may be called immediately. If you use `mysql_use_result()`, `mysql_num_rows()` will not return the correct value until all the rows in the result set have been retrieved.

20.4.36.2 Return values

The number of rows in the result set.

20.4.36.3 Errors

None.

20.4.37 `mysql_options()`

```
int mysql_options(MYSQL *mysql, enum mysql_option option, const char
*arg)
```

20.4.37.1 Description

Can be used to set extra connect options and affect behavior for a connection.

Should be called after `mysql_init()` and before `mysql_connect()` or `mysql_real_connect()`.

The `option` argument is the option that you want to set; The `arg` argument is the value for the option. If the option is an integer, then `arg` should point to the value of the integer.

Possible options values:

Option	*Argument type*	*Function*
`MYSQL_OPT_CONNECT_TIMEOUT`	unsigned int *	Connect timeout in seconds.
`MYSQL_OPT_COMPRESS`	Not used	Use the compressed client/server protocol.
`MYSQL_OPT_NAMED_PIPE`	Not used	Use named pipes to connect to a ***MySQL*** server on NT.
`MYSQL_INIT_COMMAND`	char *	Command to execute when connecting to ***MySQL*** server. Will automatically be re−executed when reconnecting.
`MYSQL_READ_DEFAULT_FILE`	char *	Read options from the named option file instead of from `` `my.cnf' ``.
`MYSQL_READ_DEFAULT_GROUP`	char *	Read options from the named group from `` `my.cnf' ``.or the file specified with `MYSQL_READ_DEFAULT_FILE`.

Note that the group `client` is always read if you use `MYSQL_READ_DEFAULT_FILE` or `MYSQL_READ_DEFAULT_GROUP`.

The specified group in the option file may contain the following options:

compress	Use the compressed client/server protocol.
database	Connect to this database if there was no database in the connect command
debug	Debug options
host	Default host name
init−command	Command to execute when connecting to ***MySQL*** server. Will automatically be re−executed when reconnecting.
password	Default password
pipe	Use named pipes to connect to a ***MySQL*** server on NT.
port	Default port number
return−found−rows	Tell `mysql_info()` to return found rows instead of updated rows when using `UPDATE`.
socket	Default socket number
timeout	Connect timeout in seconds.
user	Default user

For more information about option files, see 4.15.4 Fichier d'options.

20.4.37.2 Return values

Zero for success. Non−zero if you used an unknown option.

20.4.37.3 Example

```
MYSQL mysql;

mysql_init(;
mysql_options(MYSQL_OPT_COMPRESS,0);
mysql_options(MYSQL_READ_DEFAULT_GROUP,"odbc");
if (!mysql_real_connect(38;mysql,"host","user","passwd","database",0,NULL,0))
{
    fprintf(stderr, "Failed to connect to database: Error: %s\n",
          mysql_error();
}
```

The above requests the client to use the compressed client/server protocol and read the additional options from the `odbc` section in the `my.cnf` file.

20.4.38 `mysql_ping()`

```
int mysql_ping(MYSQL *mysql)
```

20.4.38.1 Description

Checks whether or not the connection to the server is working. If it has gone down, an automatic reconnection is attempted.

This function can be used by clients that remain idle for a long while, to check whether or not the server has closed the connection and reconnect if necessary.

20.4.38.2 Return values

Zero if the server is alive. Non−zero if an error occurred.

20.4.38.3 Errors

CR_COMMANDS_OUT_OF_SYNC
: Commands were executed in an improper order.

CR_SERVER_GONE_ERROR
: The ***MySQL*** server has gone away.

CR_UNKNOWN_ERROR
: An unknown error occurred.

20.4.39 mysql_query()

```
int mysql_query(MYSQL *mysql, const char *query)
```

20.4.39.1 Description

Executes the SQL query pointed to by the null−terminated string `query`. The query must consist of a single SQL statement. You should not add a terminating semicolon (`;') or `\g` to the statement.

`mysql_query()` cannot be used for requêtesthat contain binary data; you should use `mysql_real_query()` instead. (Binary data may contain the `\0' character, which `mysql_query()` interprets as the end of the query string.)

20.4.39.2 Return values

Zero if the query was successful. Non−zero if an error occurred.

20.4.39.3 Errors

CR_COMMANDS_OUT_OF_SYNC
: Commands were executed in an improper order.

CR_SERVER_GONE_ERROR
: The ***MySQL*** server has gone away.

CR_SERVER_LOST
: The connection to the server was lost during the query.

CR_UNKNOWN_ERROR
: An unknown error occurred.

20.4.40 mysql_real_connect()

```
MYSQL *mysql_real_connect(MYSQL *mysql, const char *host, const char
*user, const char *passwd, const char *db, unsigned int port, const char
*unix_socket, unsigned int client_flag)
```

20.4.40.1 Description

`mysql_real_connect()` attempts to establish a connection to a ***MySQL*** database engine running on `host`. `mysql_real_connect()` must complete successfully before you can execute any of the other API functions, with the exception of `mysql_get_client_info()`.

The parameters are specified as follows:

- `mysql` is a pointer to a `MYSQL` connection structure, or `NULL`. If `mysql` is `NULL`, the C API allocates memory for the connection structure automatically and frees it when you call `mysql_close()`. The disadvantage of this approach is that you can't retrieve an error message if the connection fails. (To get error information from `mysql_errno()` or `mysql_error()`, you must provide a valid `MYSQL` pointer.) If the first parameter is not `NULL`, it should be the address of an existing `MYSQL` structure. In this case, before calling `mysql_real_connect()` you must call `mysql_init()` to initialize the `MYSQL` structure. See the example below.
- The value of `host` may be either a hostname or an IP address. If `host` is `NULL` or the string `"localhost"`, a connection to the local host is assumed. If the OS supports sockets (Unix) or named pipes (Win32), they are used instead of TCP/IP to connect to the server.
- The `user` parameter contains the user's ***MySQL*** login ID. If `user` is `NULL`, the current user is assumed. Under Unix, this is the current login name. Under Windows ODBC, the current user name must be specified explicitly. 16.4 Comment remplir les différents champs du gestionnaire ODBC.
- The `passwd` parameter contains the password for `user`. If `passwd` is `NULL`, only entries in the `user` table for the user that have a blank password field will be checked for a match. This allows the database administrator to set up the ***MySQL*** privilege system in such a way that users get different privileges depending on whether or not they have specified a password. Note: Do not attempt to encrypt the password before calling `mysql_real_connect()`; password encryption is handled automatically by the client API.
- `db` is the database name. If `db` is not `NULL`, the connection will set the default database to this value.
- If `port` is not 0, the value will be used as the port number for the TCP/IP connection. Note that the `host` parameter determines the type of the connection.
- If `unix_socket` is not `NULL`, the string specifies the socket or named pipe that should be used. Note that the `host` parameter determines the type of the connection.
- The value of `client_flag` is usually 0, but can be set to a combination of the following flags in very special circumstances:

Flag name	***Flag meaning***
`CLIENT_FOUND_ROWS`	Return the number of found rows, not the number of affected rows
`CLIENT_NO_SCHEMA`	Don't allow the `nom_base_de_donnees.nom_table.nom_colonne` syntax. This is for ODBC; it causes the parser to generate an error if you use that syntax, which is is useful for trapping bugs in some ODBC programs.
`CLIENT_COMPRESS`	Use compression protocol
`CLIENT_ODBC`	The client is an ODBC client. This changes `mysqld` to be more ODBC−friendly.

20.4.40.2 Return values

A `MYSQL*` connection handle if the connection was successful. `NULL` if the connection was unsuccessful. For a successful connection, the return value is the same as the value of the first parameter, unless you pass `NULL` for that parameter.

20.4.40.3 Errors

`CR_CONN_HOST_ERROR`
: Failed to connect to the ***MySQL*** server.

`CR_CONNECTION_ERROR`
: Failed to connect to the local ***MySQL*** server.

`CR_IPSOCK_ERROR`
: Failed to create an IP socket.

`CR_OUT_OF_MEMORY`
: Out of memory.

`CR_SOCKET_CREATE_ERROR`
: Failed to create a Unix socket.

`CR_UNKNOWN_HOST`
: Failed to find the IP address for the hostname.

`CR_VERSION_ERROR`
: A protocol mismatch resulted from attempting to connect to a server with a client library that uses a different protocol version. This can happen if you use a very old client library to connect to a new server that wasn't started with the `--old-protocol` option.

`CR_NAMEDPIPEOPEN_ERROR;`
: Failed to create a named pipe on Win32.

`CR_NAMEDPIPEWAIT_ERROR;`
: Failed to wait for a named pipe on Win32.

`CR_NAMEDPIPESETSTATE_ERROR;`
: Failed to get a pipe handler on Win32.

20.4.40.4 Example

```
MYSQL mysql;

mysql_init(;
if (!mysql_real_connect(38;mysql,"host","user","passwd","database",0,NULL,0))
{
    fprintf(stderr, "Failed to connect to database: Error: %s\n",
          mysql_error();
}
```

20.4.41 mysql_real_query()

```
int mysql_real_query(MYSQL *mysql, const char *query, unsigned int
length)
```

20.4.41.1 Description

Executes the SQL query pointed to by query, which should be a string `length` bytes long. The query must consist of a single SQL statement. You should not add a terminating semicolon (`;') or `\g` to the statement.

You *must* use `mysql_real_query()` rather than `mysql_query()` for requêtesthat contain binary data, since binary data may contain the `\0' character. In addition, `mysql_real_query()` is faster than `mysql_query()` since it does not call `strlen()` on the query string.

20.4.41.2 Return values

Zero if the query was successful. Non−zero if an error occurred.

20.4.41.3 Errors

CR_COMMANDS_OUT_OF_SYNC
: Commands were executed in an improper order.

CR_SERVER_GONE_ERROR
: The ***MySQL*** server has gone away.

CR_SERVER_LOST
: The connection to the server was lost during the query.

CR_UNKNOWN_ERROR
: An unknown error occurred.

20.4.42 mysql_reload()

```
int mysql_reload(MYSQL *mysql)
```

20.4.42.1 Description

Asks the ***MySQL*** server to reload the grant tables. The connected user must have the ***reload*** privilege.

This function is deprecated. It is preferable to use `mysql_query()` to issue a SQL `FLUSH PRIVILEGES` statement instead.

20.4.42.2 Return values

Zero for success. Non−zero if an error occurred.

20.4.42.3 Errors

CR_COMMANDS_OUT_OF_SYNC
: Commands were executed in an improper order.

CR_SERVER_GONE_ERROR
: The ***MySQL*** server has gone away.

CR_SERVER_LOST
: The connection to the server was lost during the query.

CR_UNKNOWN_ERROR
: An unknown error occurred.

20.4.43 `mysql_row_seek()`

```
MYSQL_ROW_OFFSET mysql_row_seek(MYSQL_RES *result, MYSQL_ROW_OFFSET
offset)
```

20.4.43.1 Description

Sets the row cursor to an arbitrary row in a query result set. This requires that the result set structure contains the entire result of the query, so `mysql_row_seek()` may be used in conjunction only with `mysql_store_result()`, not with `mysql_use_result()`.

The offset should be a value returned from a call to `mysql_row_tell()` or to `mysql_row_seek()`. This value is not simply a row number; if you want to seek to a row within a result set using a row number, use `mysql_data_seek()` instead.

20.4.43.2 Return values

The previous value of the row cursor. This value may be passed to a subsequent call to `mysql_row_seek()`.

20.4.43.3 Errors

None.

20.4.44 `mysql_row_tell()`

```
MYSQL_ROW_OFFSET mysql_row_tell(MYSQL_RES *result)
```

20.4.44.1 Description

Returns the current position of the row cursor for the last `mysql_fetch_row()`. This value can be used as an argument to `mysql_row_seek()`.

You should use `mysql_row_tell()` only after `mysql_store_result()`, not after `mysql_use_result()`.

20.4.44.2 Return values

The current offset of the row cursor.

20.4.44.3 Errors

None.

20.4.45 mysql_select_db()

```
int mysql_select_db(MYSQL *mysql, const char *db)
```

20.4.45.1 Description

Causes the database specified by `db` to become the default (current) database on the connection specified by `mysql`. In subsequent queries, this database is the default for table references that do not include an explicit database specifier.

`mysql_select_db()` fails unless the connected user can be authenticated as having permission to use the database.

20.4.45.2 Return values

Zero for success. Non−zero if an error occurred.

20.4.45.3 Errors

CR_COMMANDS_OUT_OF_SYNC
: Commands were executed in an improper order.

CR_SERVER_GONE_ERROR
: The ***MySQL*** server has gone away.

CR_SERVER_LOST
: The connection to the server was lost during the query.

CR_UNKNOWN_ERROR
: An unknown error occurred.

20.4.46 mysql_shutdown()

```
int mysql_shutdown(MYSQL *mysql)
```

20.4.46.1 Description

Asks the database server to shutdown. The connected user must have ***shutdown*** privileges.

20.4.46.2 Return values

Zero for success. Non−zero if an error occurred.

20.4.46.3 Errors

CR_COMMANDS_OUT_OF_SYNC
: Commands were executed in an improper order.

CR_SERVER_GONE_ERROR
: The ***MySQL*** server has gone away.

CR_SERVER_LOST
: The connection to the server was lost during the query.

CR_UNKNOWN_ERROR
: An unknown error occurred.

20.4.47 mysql_stat()

```
char *mysql_stat(MYSQL *mysql)
```

20.4.47.1 Description

Returns a character string containing information similar to that provided by the `mysqladmin status` command. This includes uptime in seconds and the number of running threads, questions, reloads and open tables.

20.4.47.2 Return values

A character string describing the server status. `NULL` if an error occurred.

20.4.47.3 Errors

CR_COMMANDS_OUT_OF_SYNC
: Commands were executed in an improper order.

CR_SERVER_GONE_ERROR
: The ***MySQL*** server has gone away.

CR_SERVER_LOST
: The connection to the server was lost during the query.

CR_UNKNOWN_ERROR
: An unknown error occurred.

20.4.48 mysql_store_result()

```
MYSQL_RES *mysql_store_result(MYSQL *mysql)
```

20.4.48.1 Description

You must call `mysql_store_result()` or `mysql_use_result()` for every query which successfully retrieves data (`SELECT`, `SHOW`, `DESCRIBE`, `EXPLAIN`).

`mysql_store_result()` reads the entire result of a query to the client, allocates a `MYSQL_RES` structure, and places the result into this structure.

An empty result set is returned if there are no rows returned. (An empty result set differs from a `NULL` return value.)

Once you have called `mysql_store_result()`, you may call `mysql_num_rows()` to find out how many rows are in the result set.

You can call `mysql_fetch_row()` to fetch rows from the result set, or `mysql_row_seek()` and `mysql_row_tell()` to obtain or set the current row position within the result set.

You must call `mysql_free_result()` once you are done with the result set.

`NULL mysql store result()`.

20.4.48.2 Return values

A `MYSQL_RES` result structure with the results. `NULL` if an error occurred.

20.4.48.3 Errors

`CR_COMMANDS_OUT_OF_SYNC`
: Commands were executed in an improper order.

`CR_OUT_OF_MEMORY`
: Out of memory.

`CR_SERVER_GONE_ERROR`
: The ***MySQL*** server has gone away.

`CR_SERVER_LOST`
: The connection to the server was lost during the query.

`CR_UNKNOWN_ERROR`
: An unknown error occurred.

20.4.49 `mysql_thread_id()`

```
unsigned long mysql_thread_id(MYSQL *mysql)
```

20.4.49.1 Description

Returns the thread ID of the current connection. This value can be used as an argument to `mysql_kill()` to kill the thread.

If the connection is lost and you reconnect with `mysql_ping()`, the thread ID will change. This means you should not get the thread ID and store it for later, you should get it when you need it.

20.4.49.2 Return values

The thread ID of the current connection.

20.4.49.3 Errors

None.

20.4.50 `mysql_use_result()`

```
MYSQL_RES *mysql_use_result(MYSQL *mysql)
```

20.4.50.1 Description

You must call `mysql_store_result()` or `mysql_use_result()` for every query which successfully retrieves data (`SELECT`, `SHOW`, `DESCRIBE`, `EXPLAIN`).

`mysql_use_result()` initiates a result set retrieval but does not actually read the result set into the client like `mysql_store_result()` does. Instead, each row must be retrieved individually by making calls to `mysql_fetch_row()`. This reads the result of a query directly from the server without storing it in a temporary table or local buffer, which is somewhat faster and uses much less memory than `mysql_store_result()`. The client will only allocate memory for the current row and a communication buffer that may grow up to `max_allowed_packet` bytes.

On the other hand, you shouldn't use `mysql_use_result()` if you are doing a lot of processing for each row on the client side, or if the output is sent to a screen on which the user may type a `^S` (stop scroll). This will tie up the server and prevent other threads from updating any tables from which the data are fetched.

When using `mysql_use_result()`, you must execute `mysql_fetch_row()` until a `NULL` value is returned, otherwise the unfetched rows will be returned as part of the result set for your next query. The C API will give the error `Commands out of sync; You can't run this command now` if you forget to do this!

You may not use `mysql_data_seek()`, `mysql_row_seek()`, `mysql_row_tell()`, `mysql_num_rows()` or `mysql_affected_rows()` with a result returned from `mysql_use_result()`, nor may you issue other requêtesuntil the `mysql_use_result()` has finished. (However, after you have fetched all the rows, `mysql_num_rows()` will accurately return the number of rows fetched.)

You must call `mysql_free_result()` once you are done with the result set.

20.4.50.2 Return values

A `MYSQL_RES` result structure. `NULL` if an error occurred.

20.4.50.3 Errors

`CR_COMMANDS_OUT_OF_SYNC`
: Commands were executed in an improper order.

`CR_OUT_OF_MEMORY`
: Out of memory.

`CR_SERVER_GONE_ERROR`
: The ***MySQL*** server has gone away.

`CR_SERVER_LOST`
: The connection to the server was lost during the query.

`CR_UNKNOWN_ERROR`
: An unknown error occurred.

20.4.51 Pourquoi après un `mysql_query()` réussi, `mysql_store_result()` retourne parfois `NULL`?

It is possible for `mysql_store_result()` to return `NULL` following a successful call to `mysql_query()`. When this happens, it means one of the following conditions occurred:

- There was a `malloc()` failure (for example, if the result set was too large).

- The data couldn't be read (an error occurred on the connection).
- The query returned no data (e.g., it was an INSERT, UPDATE or DELETE).

You can always check whether or not the statement should have produced a non−empty result by calling `mysql_field_count()`. If `mysql_field_count()` returns zero, the result is empty and the last query was a statement that does not return values (for example, an INSERT or a DELETE). If `mysql_field_count()` returns a non−zero value, the statement should have produced a non−empty result. See the description of the `mysql_field_count()` function for an example.

You can test for an error by calling `mysql_error()` or `mysql_errno()`.

20.4.52 Quels sont les résultats à attendre d'une requête

In addition to the result set returned by a query, you can also get the following information:

- `mysql_affected_rows()` returns the number of rows affected by the last query when doing an INSERT, UPDATE or DELETE. An exception is that if DELETE is used without a WHERE clause, the table is truncated, which is much faster! In this case, `mysql_affected_rows()` returns zero for the number of records affected.
- `mysql_num_rows()` returns the number of rows in a result set. With `mysql_store_result()`, `mysql_num_rows()` may be called as soon as `mysql_store_result()` returns. With `mysql_use_result()`, `mysql_num_rows()` may be called only after you have fetched all the rows with `mysql_fetch_row()`.
- `mysql_insert_id()` returns the ID generated by the last query that inserted a row into a table with an AUTO_INCREMENT index. mysql_insert_id().
- Some requêtes(`LOAD DATA INFILE ...`, `INSERT INTO ... SELECT ...`, UPDATE) return additional info. The result is returned by `mysql_info()`. See the description for `mysql_info()` for the format of the string that it returns. `mysql_info()` returns a NULL pointer if there is no additional information.

20.4.53 Comment obtenir l'ID unique de la dernière ligne insérée?

If you insert a record in a table containing a column that has the AUTO_INCREMENT attribute, you can get the most recently generated ID by calling the `mysql_insert_id()` function.

You can also retrieve the ID by using the `LAST_INSERT_ID()` function in a query string that you pass to `mysql_query()`.

You can check if an AUTO_INCREMENT index is used by executing the following code. This also checks if the query was an INSERT with an AUTO_INCREMENT index:

```
if (mysql_error(38;mysql)[0] == 0 38;    mysql_num_fields(result) == 0 38;    mysql_insert_id(38;
{
    used_id = mysql_insert_id(;
}
```

The most recently generated ID is maintained in the server on a per−connection basis. It will not be changed by another client. It will not even be changed if you update another AUTO_INCREMENT column with a non−magic value (that is, a value that is not NULL and not 0).

If you want to use the ID that was generated for one table and insert it into a second table, you can use SQL statements like this:

```
INSERT INTO foo (auto,text)
    VALUES(NULL,'text');              # generate ID by inserting NULL
INSERT INTO foo2 (id,text)
    VALUES(LAST_INSERT_ID(),'text');  # use ID in second table
```

20.4.54 Problème de link avec les API C

When linking with the C API, the following errors may occur on some systems:

```
gcc −g −o client test.o −L/usr/local/lib/mysql −lmysqlclient −lsocket −lnsl

Undefined        first referenced
 symbol          in file
floor            /usr/local/lib/mysql/libmysqlclient.a(password.o)
ld: fatal: Symbol referencing errors. No output written to client
```

If this happens on your system, you must include the math library by adding `−lm` to the end of the compile/link line.

20.4.55 Comment rendre le client thread−safe

The client is ``almost'' thread−safe. The biggest problem is that the subroutines in `net.c' that read from sockets are not interrupt−safe. This was done with the thought that you might want to have your own alarm that can break a long read to a server.

The standard client libraries are not compiled with the thread options.

To get a thread−safe client, use the `−lmysys`, `−lstring` and `−ldbug` libraries and `net_serv.o` that the server uses.

When using a threaded client, you can make great use of the routines in the `thr_alarm.c' file. If you are using routines from the `mysys` library, the only thing you must remember is to call `my_init()` first!

All functions except `mysql_real_connect()` are currently thread−safe. The following notes describe how to compile a thread−safe client library and use it in a thread−safe manner. (The notes below for `mysql_real_connect()` actually apply to `mysql_connect()` as well, but since `mysql_connect()` is deprecated, you should be using `mysql_real_connect()` anyway.)

To make `mysql_real_connect()` thread−safe, you must recompile the client library with this command:

```
shell62; CPPFLAGS=−DTHREAD_SAFE_CLIENT ./configure ...
```

You may get some errors because of undefined symbols when linking the standard client, because the pthread libraries are not included by default.

The resulting `libmysqlclient.a' library is now thread−safe. What this means is that client code is thread−safe as long as two threads don't query the same connection handle returned by `mysql_real_connect()` at the same time; the client/server protocol allows only one request at a time on a given connection. If you want to use multiple threads on the same connection, you must have a mutex lock around your `mysql_query()` and `mysql_store_result()` call combination. Once `mysql_store_result()` is ready, the lock can be released and other threads may query the same connection. (In other words, different threads can use different `MYSQL_RES` pointers that were created with `mysql_store_result()`, as long as they use the proper locking protocol.) If you program with POSIX threads, you can use `pthread_mutex_lock()` and `pthread_mutex_unlock()` to establish and release a mutex lock.

If you used `mysql_use_result()` rather than `mysql_store_result()`, the lock would need to surround `mysql_use_result()` and the calls to `mysql_fetch_row()`. However, it really is best for threaded clients not to use `mysql_use_result()`.

20.5 API MySQL en Perl

This section documents the Perl `DBI` interface. The former interface was called `mysqlperl`. Since `DBI`/`DBD` now is the recommended Perl interface, `mysqlperl` is obsolete and is not documented here.

20.5.1 `DBI` avec `DBD::mysql`

`DBI` is a generic interface for many databases. That means that you can write a script that works with many different database engines without change. You need a DataBase Driver (DBD) defined for each database type. For ***MySQL***, this driver is called `DBD::mysql`.

For more information on the Perl5 DBI, please visit the `DBI` web page and read the documentation:

`http://www.symbolstone.org/technology/perl/DBI/index.html`

For more information on Object Oriented Programming (OOP) as defined in Perl5, see the Perl OOP page:

`http://language.perl.com/info/documentation.html`

Installation instructions for ***MySQL*** Perl support are given in 4.10 Remarques sur l'installation Perl.

20.5.2 L'interface `DBI`

Portable DBI méthodes

`connect`	Establishes a connection to a database server
`disconnect`	Disconnects from the database server
`prepare`	Prepares a SQL statement for execution
`execute`	Executes prepared statements
`do`	Prepares and executes a SQL statement
`quote`	Quotes string or `BLOB` values to be inserted
`fetchrow_array`	Fetches the next row as an array of fields.
`fetchrow_arrayref`	Fetches next row as a reference array of fields
`fetchrow_hashref`	Fetches next row as a reference to a hashtable
`fetchall_arrayref`	Fetches all data as an array of arrays
`finish`	Finishes a statement and let the system free resources
`rows`	Returns the number of rows affected
`data_sources`	Returns an array of databases available on localhost
`ChopBlanks`	Controls whether `fetchrow_*` méthodes trim spaces
`NUM_OF_PARAMS`	The number of placeholders in the prepared statement
`NULLABLE`	Which columns can be `NULL`
`trace`	Perform tracing for debugging

MySQL−specific méthodes

`insertid`	The latest `AUTO_INCREMENT` value
`is_blob`	Which column are `BLOB` values
`is_key`	Which columns are keys
`is_num`	Which columns are numeric
`is_pri_key`	Which columns are primary keys
`is_not_null`	Which columns CANNOT be `NULL`. See `NULLABLE`.
`length`	Maximum possible column sizes
`max_length`	Maximum column sizes actually present in result
`NAME`	Column names
`NUM_OF_FIELDS`	Number of fields returned
`table`	Table names in returned set
`type`	All column types

The Perl méthodes are described in more detail in the following sections. Variables used for méthode return values have these meanings:

`$dbh`
: Database handle

`$sth`
: Statement handle

`$rc`
: Return code (often a status)

`$rv`
: Return value (often a row count)

Portable DBI méthodes

`connect($data_source, $username, $password)`

Use the `connect` méthode to make a database connection to the data source. The `$data_source` value should begin with `DBI:driver_name:`. Example uses of `connect` with the `DBD::mysql` driver:

```
$dbh = DBI−62;connect("DBI:mysql:$database", $user, $password);
$dbh = DBI−62;connect("DBI:mysql:$database:$hostname",
                      $user, $password);
$dbh = DBI−62;connect("DBI:mysql:$database:$hostname:$port",
                      $user, $password);
```

If the user name and/or password are undefined, `DBI` uses the values of the `DBI_USER` and `DBI_PASS` environment variables, respectively. If you don't specify a hostname, it defaults to `'localhost'`. If you don't specify a port number, it defaults to the default ***MySQL*** port (33). As of `Msql−Mysql−modules` version 1.2009, the `$data_source` value allows certain modifiers:

`mysql_read_default_file=file_name`
: Read `` `filename' `` as an option file. For information on option files, see 4.15.4 Fichier d'options.

`mysql_read_default_group=group_name`
: The default group when reading an option file is normally the `[client]` group. By specifying the `mysql_read_default_group` option, the default group becomes the `[group_name]` group.

`mysql_compression=1`
: Use compressed communication between the client and server (***MySQL*** 3.22.3 or later).

`mysql_socket=/path/to/socket`
: Specify the pathname of the Unix socket that is used to connect to the server (***MySQL*** 3.21.15 or later).

Multiple modifiers may be given; each must be preceded by a semicolon. For example, if you want to avoid hardcoding the user name and password into a `DBI` script, you can take them from the user's `` `~/.my.cnf' `` option file instead by writing your `connect` call like this:

```
$dbh = DBI−62;connect("DBI:mysql:$database"
                . ";mysql_read_default_file=$ENV{HOME}/.my.cnf",
                $user, $password);
```

This call will read options defined for the `[client]` group in the option file. If you wanted to do the same thing, but use options specified for the `[perl]` group instead, you could use this:

```
$dbh = DBI−62;connect("DBI:mysql:$database"
                . ";mysql_read_default_file=$ENV{HOME}/.my.cnf"
                . ";mysql_read_default_group=perl",
                $user, $password);
```

disconnect

The `disconnect` méthode disconnects the database handle from the database. This is typically called right before you exit from the program. Example:

```
$rc = $dbh−62;disconnect;
```

prepare($statement)

Prepares a SQL statement for execution by the database engine and returns a statement handle `($sth)` which you can use to invoke the `execute` méthode. Typically you handle `SELECT` statements (and `SELECT`−like statements such as `SHOW`, `DESCRIBE` and `EXPLAIN`) by means of `prepare` and `execute`. Example:

```
$sth = $dbh−62;prepare($statement)
    or die "Can't prepare $statement: $dbh−62;errstr\n";
```

execute

The `execute` méthode executes a prepared statement. For non−`SELECT` statements, `execute` returns the number of rows affected. If no rows are affected, `execute` returns `"0E0"`, which Perl treats as zero but regards as true. For `SELECT` statements, `execute` only starts the SQL query in the database; you need to use one of the `fetch_*` méthodes described below to retrieve the data. Example:

```
$rv = $sth−62;execute
          or die "can't execute the query: $sth−62;errstr;
```

do($statement)

The `do` méthode prepares and executes a SQL statement and returns the number of rows affected. If no rows are affected, `do` returns `"0E0"`, which Perl treats as zero but regards as true. This méthode is generally used for non−`SELECT` statements which cannot be prepared in advance (due to driver limitations) or which do not need to executed more than once (inserts, deletes, etc.). Example:

```
$rv = $dbh−62;do($statement)
        or die "Can't execute $statement: $dbh− 62;errstr\n";
```

quote($string)

The `quote` méthode is used to "escape" any special characters contained in the string and to add the required outer quotation marks. Example:

```
$sql = $dbh−62;quote($string)
```

fetchrow_array

This méthode fetches the next row of data and returns it as an array of field values. Example:

```
while(@row = $sth−62;fetchrow_array) {
        print qw($row[0]\t$row[1]\t$row[2]\n);
}
```

fetchrow_arrayref

This méthode fetches the next row of data and returns it as a reference to an array of field values. Example:

```
while($row_ref = $sth−62;fetchrow_arrayref) {
        print qw($row_ref−62;[0]\t$row_ref−62;[1]\t$row_ref−62;[2]\n);
}
```

fetchrow_hashref

This méthode fetches a row of data and returns a reference to a hash table containing field name/value pairs. This méthode is not nearly as

efficient as using array references as demonstrated above. Example:

```
while($hash_ref = $sth−62;fetchrow_hashref) {
        print qw($hash_ref−62;{firstname}\t$hash_ref−62;{lastname}\t\
                $hash_ref− 62; title}\n);
}
```

fetchall_arrayref

This méthode is used to get all the data (rows) to be returned from the SQL statement. It returns a reference to an array of arrays of references to each row. You access or print the data by using a nested loop. Example:

```
my $table = $sth−62;fetchall_arrayref
                or die "$sth−62;errstr\n";
my($i, $j);
for $i ( 0 .. $#{$table} ) {
        for $j ( 0 .. $#{$table−62;[$i]} ) {
                print "$table−62;[$i][$j]\t";
        }
        print "\n";
}
```

finish

Indicates that no more data will be fetched from this statement handle. You call this méthode to free up the statement handle and any system resources it may be holding. Example:

```
$rc = $sth−62;finish;
```

rows

Returns the number of rows changed (updated, deleted, etc.) by the last command. This is usually used after a non−`SELECT execute` statement. Example:

```
$rv = $sth−62;rows;
```

NULLABLE

Returns a reference to an array of boolean values; for each element of the array, a value of TRUE indicates that this column may contain `NULL` values. Example:

```
$null_possible = $sth−62;{NULLABLE};
```

NUM_OF_FIELDS

This attribute indicates the number of fields returned by a `SELECT` or `SHOW FIELDS` statement. You may use this for checking whether a statement returned a result: A zero value indicates a non−`SELECT` statement like `INSERT`, `DELETE` or `UPDATE`. Example:

```
$nr_of_fields = $sth−62;{NUM_OF_FIELDS};
```

data_sources($driver_name)

This méthode returns an array containing names of databases available to the ***MySQL*** server on the host `'localhost'`. Example:

```
@dbs = DBI−62;data_sources("mysql");
```

ChopBlanks

This attribute determines whether the `fetchrow_*` méthodes will chop leading and trailing blanks from the returned values. Example:

```
$sth−62;{'ChopBlanks'} =1;
```

trace($trace_level)
trace($trace_level, $trace_filename)

The `trace` méthode enables or disables tracing. When invoked as a `DBI` class méthode, it affects tracing for all handles. When invoked as a database or statement handle méthode, it affects tracing for the given handle (and any future children of the handle). Setting `$trace_level` to 2 provides detailed trace information. Setting `$trace_level` to 0 disables tracing. Trace output goes to the standard error output by default. If `$trace_filename` is specified, the file is opened in append mode and output for *all* traced handles is written to that file. Example:

```
DBI−62;trace(2);                # trace everything
DBI−62;trace(2,"/tmp/dbi.out"); # trace everything to /tmp/dbi.out
$dth−62;trace(2);               # trace this database handle
$sth−62;trace(2);               # trace this statement handle
```

You can also enable `DBI` tracing by setting the `DBI_TRACE` environment variable. Setting it to a numeric value is equivalent to calling `DBI−>(value)`. Setting it to a pathname is equivalent to calling `DBI−>(2,value)`.

MySQL−specific méthodes

The méthodes shown below are ***MySQL***−specific and not part of the `DBI` standard. Several of them are now deprecated: `is_blob`, `is_key`, `is_num`, `is_pri_key`, `is_not_null`, `length`, `max_length`, and `table`. Where `DBI`−standard alternatives exist, they are noted below.

`insertid`

If you use the `AUTO_INCREMENT` feature of ***MySQL***, the new auto−incremented values will be stored here. Example:

```
$new_id = $sth−62;{insertid};
```

As an alternative, you can use `$dbh−>{'mysql_insertid'}`.

`is_blob`

Returns a reference to an array of boolean values; for each element of the array, a value of TRUE indicates that the respective column is a `BLOB`. Example:

```
$keys = $sth−62;{is_blob};
```

`is_key`

Returns a reference to an array of boolean values; for each element of the array, a value of TRUE indicates that the respective column is a key. Example:

```
$keys = $sth−62;{is_key};
```

`is_num`

Returns a reference to an array of boolean values; for each element of the array, a value of TRUE indicates that the respective column contains numeric values. Example:

```
$nums = $sth−62;{is_num};
```

`is_pri_key`

Returns a reference to an array of boolean values; for each element of the array, a value of TRUE indicates that the respective column is a primary key. Example:

```
$pri_keys = $sth−62;{is_pri_key};
```

`is_not_null`

Returns a reference to an array of boolean values; for each element of the array, a value of FALSE indicates that this column may contain `NULL` values. Example:

```
$not_nulls = $sth−62;{is_not_null};
```

`is_not_null` is deprecated; it is preferable to use the `NULLABLE` attribute (described above), since that is a DBI standard.

`length`
`max_length`

Each of these méthodes returns a reference to an array of column sizes. The `length` array indicates the maximum possible sizes that each column may be (as declared in the table description). The `max_length` array indicates the maximum sizes actually present in the result table. Example:

```
$lengths = $sth−62;{length};
$max_lengths = $sth−62;{max_length};
```

NAME
: Returns a reference to an array of column names. Example:
```
$names = $sth−62;{NAME};
```

table
: Returns a reference to an array of table names. Example:
```
$tables = $sth−62;{table};
```

type
: Returns a reference to an array of column types. Example:
```
$types = $sth−62;{type};
```

20.5.3 Plus d'informations sur DBI/DBD

You can use the `perldoc` command to get more information about `DBI`.

```
perldoc DBI
perldoc DBI::FAQ
perldoc DBD::mysql
```

You can also use the `pod2man`, `pod2html`, etc., tools to translate to other formats.

And of course you can find the latest `DBI` information at the `DBI` web page:

`http://www.symbolstone.org/technology/perl/DBI/index.html`

20.6 MySQL Eiffel

The ***MySQL*** Contrib directory contains an Eiffel wrapper written by Michael Ravits.

You can also find this at: http://www.netpedia.net/hosting/newplayer/

20.7 Connexion Java MySQL (ODBC)

There are 2 supported JDBC drivers for MySQL (the twz and mm driver). You can find a copy of these at http://www.mysql.com/Contrib. For documentation consult any JDBC documentation and the drivers own documentation for ***MySQL*** specific features.

20.8 API MySQL PHP

PHP is a server−side, HTML embedded scripting language that may be used to create dynamic web pages. It contains support for accessing several databases, including ***MySQL***. PHP may be run as a separate program, or compiled as a module for use with the Apache web server.

The distribution and documentation are available at the PHP website.

20.9 API MySQL C++

Two API's are available in the ***MySQL*** Contrib directory.

20.10 API MySQL Python

The ***MySQL*** Contrib directory contains a Python interface written by Joseph Skinner.

You can also use the Python interface to iODBC to access a ***MySQL*** server. mxODBC

20.11 API MySQL TCL

TCL at binevolve. The Contrib directory contains a TCL interface that is based on msqltcl 1.50.

21 Comparer MySQL à d'autres bases de données

21.1 Comparer MySQL et mSQL

Cette section a été écrite par des développeurs ***MySQL***, et il doit être lu avec quelques reserves. Cependant, il n'y a AUCUNE erreur factuelle, autant que nous le sachions.

Pour une liste complète de toutes les limites, fonctions et types, allez à `crash-me` (en anglais).

Performance

Pour une comparaison de performances valide, consultez la suite de tests. 11 La suite de tests de MySQL. Parce qu'il n'y a pas de création de thread général, que l'analyseur syntaxique est compact, les fonctionnalités peu nombreusese, et la sécurité simple, `mSQL` est plus rapide dans les t ches suivantes :

- ◊ Tests qui effectuent des connexions/deconnexion repetées, avec une requête tres simple entre les deux.
- ◊ Opérations `INSERT` dans des tables simples, avec quelques colonnes et clés.
- ◊ `CREATE TABLE` et `DROP TABLE`.
- ◊ `SELECT` sur ce qui n'est pas un index. (l'analyse d'une table est très simple).

Etant donné que ces opéartions sont simples, il est très difficile de les surpasser dès qu'on a une partie administrative plus importante. Apres l'établissement de la connexion, ***MySQL*** devrait être plus rapide. D'un autre coté, ***MySQL*** est beaucoup plus rapide que `mSQL` (et que la plus part des implémentation SQL) dans les cas suivants :

- ◊ Opérations `SELECT` complexes.
- ◊ Récupération de résultat très importants (***MySQL*** a un protocol pluls rapide, plus sur et bien meilleur).
- ◊ Tables à longueur d'enregistrement variable, étant donné que ***MySQL*** les gère mieux, et qu'il autorise les index sur les colonnes `VARCHAR`.
- ◊ Gestion des tables ayant beaucoup de colonnes.
- ◊ Gestion des tables ayant des lignes de très grande taille.
- ◊ `SELECT` avec de nombreuses expressions.
- ◊ `SELECT` sur les grandes tables.
- ◊ Gestion de plusieurs connexions simultanées. ***MySQL*** est complétement multi−threaded. Chaque connexion a son propre thread, ce qui fait que les threads ne s'attendent pas les uns les autres (sauf lorsque des LOCK sont posés sur les tables). Avec `mSQL`, une fois la connexion établie, les autres doivent attendre que le premier ait terminé, quelque soit la requête lancée. Lorsque cette dernière est terminée, la connexion se termine, et la prochaine connexion est ouverte.
- ◊ Jointures. `mSQL` peut devenir desespérement lent si vous changez l'ordre des tables dans la commande `SELECT`. Dans la suite de teste, il est arrivé que `mSQL` soit 15000 fois plus lent que ***MySQL*** Cela est du à `mSQL` à qui manque un optimisateur de jointures. Cependant, si vous placez les tables dans le bon ordre, si la clause `WHERE` est simple et si elle utilise des colonnes indexées, la jointure sera plutôt rapide. 11 La suite de tests de MySQL.
- ◊ `ORDER BY` et `GROUP BY`.
- ◊ `DISTINCT`.
- ◊ Utiliser des colonnes de type `TEXT` ou `BLOB`.

SQL

- ◊ `GROUP BY` and `HAVING`. `mSQL` ne supporte pas la clause `GROUP BY`. ***MySQL*** supporte complètement la clause `GROUP BY` avec la clause `HAVING` et les instructions suivantes : `COUNT()`, `AVG()`, `MIN()`, `MAX()`, `SUM()` and `STD()`. `COUNT(*)` est optimisé pour retourner TRES rapidement le bon compte si `SELECT` ne fait que compter dans une seule table, sans colonne supplémentaire, ni clause `WHERE`. `MIN()` et `MAX()` acceptent aussi des arguments chaînes.
- ◊ `INSERT` et `UPDATE` avec calculs. ***MySQL*** peut faire des calculs dans une commande `INSERT` et `UPDATE`. Par exemple :

```
mysql62; UPDATE SET x=x*10+y WHERE x60;20;
```

- ◊ Alias. ***MySQL*** permet les alias de colonnes.
- ◊ Qualification minimale des noms de colonnes. Avec ***MySQL***, si un nom de colonne est unique parmi les tables en jeu dans une requête, il n'est pas nécessaire de l'identifier avec son nom complet.
- ◊ `SELECT` avec des fonctions. ***MySQL*** dispose de nombreuses fonctions (trop nombreuses pour être listées ici : allez à 7.3 Fonctions utilisées dans les clauses `SELECT` et `WHERE`).

Utilisation du disque

A quel point pouvez vous réduire vos tables? ***MySQL*** dispose de types de colonnes précis, qui vous permettent de créer des tables qui prenent le moins de place possible. Comme exemple de type de table bien pratique, nous pouvons prendre `MEDIUMINT` qui ne prend que 3 octets. Si vous avez 100,000,000 enregistrements, économiser 25% de la place peut être primordial. `mSQL2` dispose de beaucoup moins de types de colonnes, et cela rend difficile la réduction de tables.

Stabilité

Cela est plus difficile à estimer objectivement. Pour la stabilité, voyez 1.5 Est ce que MySQL est stable?. Nous n'avons aucune experience en terme de stabilité `mSQL`, et nous ne nous étendrons pas sur ce point.

Prix

Un autre élément d'importance est la licence. ***MySQL*** a une licence plus souple que `mSQL`, et il est aussi moins cher que `mSQL`. Quelque soit le produit que vous décidez de choisir, pensez à prendre une licence ou un support email. (Il vous faut aquérir une licence pour inclure ***MySQL*** dans un produit que vous vendez).

Interfaces Perl

MySQL a pratiquement les mêmes interfaces avec Perl que `mSQL` mais avec quelques fonctionnalités en plus.

JDBC (Java)

MySQL dispose actuellement de 4 pilotes JDBC :

- ◊ Le pilote gwe : une interface Java par GWE technologies (plus du tout supportée).
- ◊ Le pilote jms : un pilote gwe amélioré par Xiaokun Kelvin ZHU
- ◊ Le pilote twz : Un pilote de type 4 JDBC par Terrence W. Zellers
- ◊ Le pilote mm : Un pilote de type 4 JDBC par Mark Matthews

Les pilotes recommendés sont twz et mm. Les deux fonctionnent parfaitement. Nous savons que `mSQL` dispose d'un pilote JDBC, mais nous n'avons que très peu d'expérience, et ne pouvons pas comparer.

Vitesse de developpement

MySQL dispose d'une petite équipe de développeur, mais nous sommes habitués à travailler en C et C++ très rapidement. Etant donné que les threads, fonctions, `GROUP BY` etc... ne sont pas implémentés dans `mSQL`, il a pas mal de retard. Pour dire les choses simplement, il suffit de comparer le fichier d'historique de `mSQL` et de le comparer à la section news de ***MySQL*** Reference Manual (D Historique des versions de MySQL). Le logiciel le plus rapidement developpé est une évidence.

Programmes utilitaires.

`mSQL` et ***MySQL*** dispose d'outils partenaires très interssants. Etant donné qu'il n'est pas facile de porter ces outils de `MySQL` à ***mSQL***), la plus part des applications interssantes, disponibles pour `mSQL` sont aussi disponibles pour ***MySQL***. ***MySQL*** est livré avec un utilitaire tout simple, `msql2mysql` qui corrige les problèmes d'orthographe dans les API C entre `mSQL` et ***MySQL*** Par exemple, il va remplacer les appels à `msqlConnect()` par `mysql_connect()`. Convertir un programme client de `mSQL` à ***MySQL*** ne prend que quelques minutes.

21.1.1 Comment convertir les outils `mSQL` pour *MySQL*

Selon notre expérience, il ne prendrait que quelques heures pour convertir des outils tels que `msql-tcl` et `msqljava` qui utilisent les API C `mSQL` pour qu'ils fonctionnent avec ***MySQL***.

Les instructions de conversion sont :

1. Exécutez le script `msql2mysql` sur les sources. Cele requiert le programme `replace`, qui est distribués par ***MySQL***.
2. Compiler.
3. Corriger les erreurs de compilation.

Les différences entre les API C `mSQL` et les API C ***MySQL*** sont :

- ***MySQL*** utilise la structure `MYSQL` comme type de connexion, (`mSQL` utilise un entier `int`).
- `mysql_connect()` prend un pointeur sur une structure `MYSQL` comme paramètre. Il est facile d'en définir un globalement, ou bien d'utiliser `malloc()` pour en créer un. `mysql_connect()` prend deux paramètres pour spécifier l'utilisateur, et le mot de passe. Vous povuez utiliser `NULL, NULL` pour appeler lesv valeurs par défaut.
- `mysql_error()` prend une structure `MYSQL` comme paramètre. Vous pouvez simplement ajouter le paramètre de votre fonction `msql_error()` si vous portez un vieux code.
- ***MySQL*** retourne une erreur et un message d'erreur, pour toutes les errerurs `mSQL` ne fait que retourner un message d'erreur.
- Quelques incompatibilités existent, notamment à cause des connexions simultanées de ***MySQL***

21.1.2 En quooi les protocoles de communications de `mSQL` et *MySQL* diffèrent?

There are enough differences that it is impossible (or at least not easy) to support both.

The most significant ways in which the ***MySQL*** protocol differs from the `mSQL` protocol are listed below:

- A message buffer may contain many result rows.
- The message buffers are dynamically enlarged if the query or the result is bigger than the current buffer, up to a configurable server and client limit.
- All packets are numbered to catch duplicated or missing packets.
- All column values are sent in ASCII. The lengths of columns and rows are sent in packed binary coding (1, 2 or 3 bytes).
- ***MySQL*** can read in the result unbuffered (without having to store the full set in the client).
- If a single write/read takes more than 30 seconds, the server closes the connection.
- If a connection is idle for 8 hours, the server closes the connection.

21.1.3 En quoi la syntaxe SQL de mSQL 2.0 diffère de *MySQL*

Column types

MySQL

Has the following additional types (among others; see CREATE TABLE):

- ◊ ENUM type for one of a set of strings.
- ◊ SET type for many of a set of strings.
- ◊ BIGINT type for 64−bit integers.

MySQL also supports the following additional type attributes:

- ◊ UNSIGNED option for integer columns.
- ◊ ZEROFILL option for integer columns.
- ◊ AUTO_INCREMENT option for integer columns that are a PRIMARY KEY. mysql_insert_id().
- ◊ DEFAULT value for all columns.

mSQL2

mSQL column types correspond to the ***MySQL*** types shown below:

mSQL *type*	*Corresponding MySQL type*
CHAR(len)	CHAR(len)
TEXT(len)	TEXT(len). len is the maximal length. And LIKE works.
INT	INT. With many more options!
REAL	REAL. Or FLOAT. Both 4− and 8−byte versions are available.
UINT	INT UNSIGNED
DATE	DATE. Uses ANSI SQL format rather than mSQL's own.
TIME	TIME
MONEY	DECIMAL(12,2). A fixed−point value with two decimals.

Index creation

MySQL

Indexes may be specified at table creation time with the CREATE TABLE statement.

mSQL

Indexes must be created after the table has been created, with separate CREATE INDEX statements.

To insert a unique identifier into a table

MySQL

Use AUTO_INCREMENT as a column type specifier. mysql_insert_id().

mSQL

Create a SEQUENCE on a table and select the _seq column.

To obtain a unique identifier for a row

MySQL

Add a PRIMARY KEY or UNIQUE key to the table.

mSQL

Use the _rowid column. Observe that _rowid may change over time depending on many factors.

To get the time a column was last modified

MySQL

Add a TIMESTAMP column to the table. This column is automatically set to the current date and time for INSERT or UPDATE statements if you don't give the column a value or if you give it a NULL value.

mSQL

Use the _timestamp column.

NULL value comparisons

MySQL
MySQL follows ANSI SQL and a comparison with `NULL` is always `NULL`.

mSQL
In `mSQL`, `NULL = NULL` is TRUE. You must change `=NULL` to `IS NULL` and `<>NULL` to `IS NOT NULL` when porting old code from `mSQL` to ***MySQL***.

String comparisons

MySQL
Normally, string comparisons are performed in case−independent fashion with the sort order determined by the current character set (ISO−8859−1 Latin1 by default). If you don't like this, declare your columns with the `BINARY` attribute, which causes comparisons to be done according to the ASCII order used on the ***MySQL*** server host.

mSQL
All string comparisons are performed in case−sensitive fashion with sorting in ASCII order.

Case−insensitive searching

MySQL
`LIKE` is a case−insensitive or case−sensitive operator, depending on the columns involved. If possible, ***MySQL*** uses indexes if the `LIKE` argument doesn't start with a wildcard character.

mSQL
Use `CLIKE`.

Handling of trailing spaces

MySQL
Strips all spaces at the end of `CHAR` and `VARCHAR` columns. Use a `TEXT` column if this behavior is not desired.

mSQL
Retains trailing space.

***WHERE* clauses**

MySQL
MySQL correctly prioritizes everything (`AND` is evaluated before `OR`). To get `mSQL` behavior in ***MySQL***, use parentheses (as shown below).

mSQL
Evaluates everything from left to right. This means that some logical calculations with more than three arguments cannot be expressed in any way. It also means you must change some requêteswhen you upgrade to ***MySQL***. You do this easily by adding parentheses. Suppose you have the following `mSQL` query:

```
mysql62; SELECT * FROM table WHERE a=1 AND b=2 OR a=3 AND b=4;
```

To make ***MySQL*** evaluate this the way that `mSQL` would, you must add parentheses:

```
mysql62; SELECT * FROM table WHERE (a=1 AND (b=2 OR (a=3 AND (b=4))));
```

Access control

MySQL
Has tables to store grant (permission) options per user, host and database. 6.6 Fonctionnement du système de droits.

mSQL
Has a file `` `mSQL.acl' `` in which you can grant read/write privileges for users.

21.2 Comparer MySQL et PostgreSQL

`PostgreSQL` has some more advanced features like user−defined types, triggers, rules and some transaction support. However, PostgreSQL lacks many of the standard types and functions from ANSI SQL and ODBC. See the `crash−me` web page for a complete list of limits and which types and functions are supported or unsupported.

Normally, `PostgreSQL` is a magnitude slower than ***MySQL***. 11 La suite de tests de MySQL. This is due

largely to their transactions system. If you really need transactions or the rich type system PostgreSQL offers and you can afford the speed penalty, you should take a look at PostgreSQL.

A Quelques utilisateurs MySQL

A.1 Sites généraux

- A pro−Linux/tech news and comment/discussion site
- All about Linux
- 32Bits Online: because there's more than one way to compute
- New about new versions of computer related stuff

A.2 Quelques moteurs de recherche

- AAA Matilda Web Search
- What's New
- Aladin
- Columbus Finder
- Spider
- Blitzsuche
- Indoseek Indonesia
- Yaboo − Yet Another BOOkmarker
- Yahoosuck
- OzSearch Internet Guide

A.3 Moteurs de recherche locaux

- Jobvertise: Post and search for jobs
- Occupational Health & Safety website databse (a project for the ECC)
- The Music Database
- Fotball (Soccer) search page
- TAKEDOWN − wrestling
- The International Lyrics Network
- Musicians looking for other musicians (Free Service)
- AddALL books searching and price comparison
- Harvard's Gray Herbarium Index of Plant Names
- Research Publications at Monash University in Australia
- The Game Development Search Engine
- My−Recipe.com; Cookbook at i−run.com
- The Innkeeper Vacation Guides
- The Mac Game Database uses PHP and MySQL

A.4 Sites Web qui utilisent MySQL comme support

- Qt Widget and Object Repository
- Brazilian samba site (in Portuguese)
- Polish General Social Survey
- Expo2000 World−wide distribution of tickets for this event is implemented using MySQL and tcl/tk. More than 5000 travel−agencies all over the world have access to it.
- FreeVote.com is a free voting service with millions of users.

A.5 Prestataires de services Internet

- Registry of Web providers that support ***MySQL***
- Dynamic DNS Services
- Dynamic domain name service
- Open DNS Project; free dynamic DNS service
- Free 3rd level domains
- Online Database
- BigBiz Internet Services
- The Virt Gazette
- Global InfoNet Inc

- WebHosters − A Guide to WWW Providers
- Internet information server
- A technology news site
- WorldNet Communications − An Internet Services Provider
- Netizen: Australian−based web consultancy
- Search site for training courses in the UK
- Gannon Chat (GPL). Written in Perl and Javascript
- A general links directory
- A web−based bookmark management service
- Walnut Creek CDROM
- WWWThreads: Interactive discussion Forums
- In Italian; Storage data from meteo station
- Online "Person To Person" Auction
- Tips on web development
- Mailfriends.com is a FREE service for everybody who wants to find friends over the internet.
- Web Page Telnet BBS List
- UniNova Digital Postcards
- DSL providers search with reviews Made with MySQL and Modperl, all pages are generated dynamically out of the MySQL database

A.6 Sites Web qui utilisent PHP et MySQL

- Jgaa's Internet − Official Support Site
- Ionline − online publication: ***MySQL***, PHP, Java, Web programming, DB development
- BaBoo(Browse and bookmark). Free web−based bookmark manager and Calendar
- Course Schedule System at Pensacola Junior College
- Florida Community College at Jacksonville
- A beginners tutoral of how to start using ***MySQL***
- 32bit.com; An extensive shareware / freeware archive
- Jokes 2000
- Burken.NU Burken is a webhotel that provides scripts, among other things, for remote users, like counters, guestbooks etc.
- tips.pair.com Contains tips on html, javascript, 2d/3d graphics and PHP3/MySQL. All pages are generated from a database.

A.7 Des consultants MySQL

- Ayni AG
- Online Database
- DataGuard (Uses ***MySQL*** and PHP)
- WWITS (Uses ***MySQL*** and PHP)
- WCN − The World Community Network
- Chip Castle Dot Com Inc
- Cybersource Pty. Ltd
- Spring infotainment gmbh & co. kg
- Develops websites using MySQL

A.8 Programmation

- The Perl CPAN Testers results page

A.9 Divers

- AZC.COM's Feature Showcase
- Course Search
- Northerbys Online Auctions
- Amsterdam Airport Schiphol
- CD database
- Used Audio Gear Database
- Musical note−sheets
- Bagism − A John Lennon fan page
- US Folk art broker
- Mail reading on the web
- Free home pages on www.somecoolname.mypage.org
- Der Server f@"ur Schulen im Web (In German)

- Auldhaefen Online Services
- CaryNET Information Center
- Dataden Computer Systems
- Andr'emuseet (In Swedish)
- HOMESITE Internet Marketing
- Jade−V Network Services
- Weather World 2010 Technical Credits
- About The Gimp plugin registry
- Java tool Archiver technical detail (Slightly optimistic about ***MySQL*** ANSI−92 compliance)
- Games Domain Cheats Database
- The "Powered By" Page (Kcilink)
- Netcasting
- NBL (Australian National Basketball League) tipping
- CGI shop
- Whirlycott: Website Design
- Museum Tusculanum Press
- Centro Siciliano di Documentazione
- Quake statistics database
- Astroforum: Astrologie and related things (in German)
- OpenDebate − Interactive Polls & Open Discussion
- Online chemical dissertation server
- FreSch! The Free Scholarship Search Service
- Stockholm Pinball Locator
- HEK A construction company
- Elsevier Bussines Information
- Medical Links (Using Coldfusion and ***MySQL***)
- Search for jobs & people at JobLink−USA
- Daily news about Linux in German language
- Competition Formation Skydiving
- E−commerce and internal accounting
- Denmark's leading business daily newspaper B@o{rsen}
- The Internet NES Database
- Travel agency in Prague in 3 languages
- Linkstation
- Searchable online database at Peoplestaff
- A searchable database system for horse classified ads
- The Poot site
- "Playin' in the LAN"; a network monitoring suite
- U.S. Army Publishing Agency
- Realestate handling in Yugoslavia
- PIMS; a Patient Information Management System
- Pilkington Software Inc
- Betazine − The Ultimate Online Beta Tester's Magazine
- A Vietnam Veteran's Memorial (The Wall) database.
- Gamer's Union specializes inauctions of used & out of print gaming material
- A daily bulletin at Monterey High school
- Computer Currents Magazine
- Community−owned site serving Lake Washington's Eastside residents and businesses
- French bowling site.

Send any additions to this list to webmaster@mysql.com.

B Contributions

Many users of ***MySQL*** have contributed *very* useful support tools and addons.

A list of what is available at http://www.mysql.com/Contrib (or any mirror) is shown below. If you want to build ***MySQL*** support for the Perl `DBI/DBD` interface, you should fetch the `Data-Dumper`, `DBI`, and `Msql-Mysql-modules` files and install them. 4.10 Remarques sur l'installation Perl.

00-README This listing.

B.1 API's

- Perl modules
  - Data-Dumper-2.09.tar.gz Perl `Data-Dumper` module. Useful with `DBI/DBD` support.
  - DBI-1.08.tar.gz Perl `DBI` module.
  - KAMXbase1.0.tar.gz Convert between `` `.dbf' `` files and ***MySQL*** tables. Perl module written by Pratap Pereira pereira@ee.eng.ohio-state.edu, extened by Kevin A. McGrail kmcgrail@digital1.peregrinehw.com. This converter can handle MEMO fields.
  - Msql-Mysql-modules-1.2206.tar.gz Perl `DBD` module to access mSQL and ***MySQL*** databases..
  - Data-ShowTable-3.3.tar.gz Perl `Data-ShowTable` module. Useful with `DBI/DBD` support.
- JDBC
  - mm.mysql.jdbc-1.0.tar.gz New java JDBC driver for ***MySQL***. This is a production release and is actively developed. By Mark Matthews (mmatthew@ecn.purdue.edu). This driver has a LGPL license. You can always find the newest driver at http://www.worldserver.com/mm.mysql/.
  - twz1jdbcForMysql-1.0.4-GA.tar.gz The twz driver: A type 4 JDBC driver by Terrence W. Zellers private and educational use.
- mysql-c++-0.02.tar.gz ***MySQL*** C++ wrapper library. By Roland Haenel,
- mysql++.1.0.tar.gz ***MySQL*** C++ API (More than just a wrapper library). Originally by
- mysql-ruby-2.1.6.tar.gz ***MySQL*** Ruby module. By TOMITA Masahiro tommy@tmtm.org Ruby is Object-Oriented Interpreter Language. ***MySQL*** C++ API (More than a wrapper library). Originally by
- delphi-interface.gz Delphi interface to `libmysql.dll`, by Blestan Tabakov,
- DelphiMySQL2.zip Delphi interface to `libmysql.dll`, by bsilva@umesd.k12.or.us
- JdmMysqlDriver-0.1.0.tar.gz A VisualWorks 3.0 Smalltalk driver for ***MySQL***. By
- Db.py Python module with caching. By gandalf@rosmail.com.
- MySQLmodule-1.4.tar.gz Python interface for the ***MySQL***. By Joseph Skinner joe@earthlight.co.nz; Modified by Joerg Senekowitsch senekow@ibm.net
- mysql_mex.tar.gz An interface program for the Matlab program by MathWorks.
- mysqltcl-1.53.tar.gz Tcl interface for ***MySQL***. Based on `` `msqltcl-1.50.tar.gz' ``. Updated by Tobias Ritzau, tobri@ida.liu.se.
- MyC-0.1.tar.gz A Visual Basic-like API, by Ed Carp.
- sqlscreens-1.0.1.tar.gz TCL/TK code to generate database screens. By Jean-Francois Dockes.
- Vdb-dflts-2.1.tar.gz This is a new version of a set of library utilities intended to provide a generic interface to SQL database engines such that your application becomes a 3-tiered application. The advantage is that you can easily switch between and move to other database engines by implementing one file for the new backend without needing to make any changes to your applications. By damian@cablenet.net.
- DbFramework-1.10.tar.gz DbFramework is a collection of classes for manipulating ***MySQL*** databases. The classes are loosely based on the CDIF Data Model Subject Area. By Paul Sharpe paul@miraclefish.com.
- pike-mysql-1.4.tar.gz ***MySQL*** module for pike. For use with the Roxen web server.
- squile.tar.gz Module for `guile` that allows `guile` to interact with SQL databases. By Hal Roberts.
- stk-mysql.tar.gz Interface for Stk. Stk is the Tk widgets with Scheme underneath instead of Tcl. By Terry Jones
- eiffel-wrapper-1.0.tar.gz. Eiffel wrapper by Michael Ravits.

B.2 Clients

- Graphical clients
  - kmysqladmin-0.32.tar.gz
  - kmysqladmin-0.3-1.src.rpm
  - kmysqladmin-0.3-1.i386.rpm An administration tool for the ***MySQL*** server using QT / KDE. Tested only on Linux.
  - Java client using Swing By Fredy Fischer, se-afs@dial.eunet.ch. Ypu can always find the latest version here.
  - mysqlwinadmn.zip Win32 GUI (binary only) to administrate a database, by David B. Mansel,
  - xmysqladmin-1.0.tar.gz A front end to the ***MySQL*** database engine. It allows reloads, status check, process control, isamchk, grant/revoke privileges, creating databases, dropping databases, create, alter, browse and drop tables. Originally by Gilbert Therrien, gilbert@ican.net but now in public domain and supported by TcX.
  - xmysql-1.9.tar.gz

    - xmysql home page A front end to the ***MySQL*** database engine. It allows for simple requêtesand table maintenance, as well as batch queries. By Rick Mehalick, dblhack@wt.net. Requires xforms 0.88 to work.
- Web clients
    - mysqladmin−atif−1.0.tar.gz WWW ***MySQL*** administrator for the `user, db` and `host` tables. By Tim Sailer, modified by Atif Ghaffar
    - mysql−webadmin−1.0a8−rz.tar.gz A tool written in PHP−FI to administrate ***MySQL*** databases remotely over the web within a Web−Browser. By Peter Kuppelwieser, Not maintained anymore!
    - mysqladm.tar.gz ***MySQL*** Web Database Administration written in Perl. By Tim Sailer.
    - mysqladm−2.tar.gz Updated version of `` `mysqladm.tar.gz' ``, by High Tide.
    - myadmin−0.4.tar.gz
    - MyAdmin home page A web based mysql administrator by Mike Machado.
    - phpMyAdmin_2.0.1.tar.gz A set of PHP3−scripts to adminstrate MySQL over the WWW.
    - phpMyAdmin home page A PHP3 tool in the spirit of mysql−webadmin, by Tobias Ratschiller, tobias@dnet.it
    - useradm.tar.gz ***MySQL*** administrator in PHP. By Ofni Thomas

B.3 Outils Web

- http://www.odbsoft.com/cook/sources.htm This package has various functions for generating html code from an SQL table structure and for generating SQL statements (Select, Insert, Update, Delete) from an html form. You can build a complete forms interface to an SQL database (query, add, update, delete) without any programming! By Marc Beneteau, marc@odbsoft.com.
- sqlhtml.tar.gz SQL/HTML is an HTML database manager for ***MySQL*** using `DBI` 1.06.
- UdmSearch−2.0.tar.gz
- UdmSearch−2.1.tar.gz A ***MySQL***− and PHP− based search engine over http. By Alexander I. Barkov bar@izhcom.ru.
- wmtcl.doc
- wmtcl.lex With this you can write HTML files with inclusions of TCL code. By
- www−sql−0.5.7.lsm
- www−sql−0.5.3.md5
- www−sql−0.5.7.tar.gz A CGI program that parses an HTML file containing special tags, parses them and inserts data from a ***MySQL*** database.
- genquery.zip Perl SQL database interface package for html.
- cgi++−0.2.tar.gz A CGI wrapper by Sasha Pachev.
- WebBoard 1.0 EU−Industries Internet−Message−Board.
- DBIx−TextIndex−0.01.tar.gz Full−text searching with Perl on `BLOB`/`TEXT` columns by Daniel Koch.

B.4 Outils d'authentification

- ascend−radius−mysql−0.4.3.patch.gz This is authentication and logging patch using ***MySQL*** for Ascend−Radius. By takeshi@SoftAgency.co.jp.
- checkpassword−0.76−mysql−0.3.2.patch.gz MySQL authentication patch for QMAIL and checkpassword. These are useful for management user(mail,pop account) by ***MySQL***. By takeshi@SoftAgency.co.jp
- jradius−diff.gz ***MySQL*** support for Livingston's Radius 2.01. Authentication and Accounting. By Jose de Leon, jdl@thevision.net
- mod_auth_mysql−2.20.tar.gz Apache authentication module for ***MySQL***. By Zeev Suraski, ***Please*** register this module at: http://bourbon.netvision.net.il/mysql/mod_auth_mysql/register.html. The registering information is only used for statistical purposes and will encourage further development of this module!
- mod_log_mysql−1.05.tar.gz ***MySQL*** logging module for Apache. By Zeev Suraski,
- mypasswd−2.0.tar.gz Extra for `mod_auth_mysql`. This is a little tool that allows you to add/change user records storing group and/or password entries in ***MySQL*** tables. By Harry Brueckner, brueckner@respublica.de.
- mysql−passwd.README
- mysql−passwd−1.2.tar.gz Extra for `mod_auth_mysql`. This is a two−part system for use with `mod_auth_mysql`.
- pam_mysql.tar.gz This module authenticates users via `pam`, using ***MySQL***.
- nsapi_auth_mysql.tar Netscape Web Server API (NSAPI) functions to authenticate (BASIC) users against ***MySQL*** tables. By Yuan John Jiang.
- qmail−1.03−mysql−0.3.2.patch.gz Patch for qmail to authenticate users from a ***MySQL*** table.
- pwcheck_mysql−0.1.tar.gz An authentication module for the Cyrus IMAP server. By Aaron Newsome.

B.5 Convertisseurs

- dbf2mysql−1.12.tar.gz Convert between `` `.dbf' `` files and ***MySQL*** tables. By Maarten Boekhold, boekhold@cindy.et.tudelft.nl, and Michael Widenius. This converter can't handle MEMO fields.
- dbf2mysql.zip Convert between FoxPro `` `.dbf' `` files and ***MySQL*** tables on Win32. By Alexander Eltsyn, ae@nica.ru or ae@usa.net.
- dump2h−1.20.gz Convert from `mysqldump` output to a C header file. By Harry Brueckner,
- exportsql.txt A script that is similar to `access_to_mysql.txt`, except that this one is fully configurable, has better type conversion (including detection of `TIMESTAMP` fields), provides warnings and suggestions while converting, quotes ***all*** special characters in text and binary data, and so on. It will also convert to `mSQL` v1 and v2, and is free of charge for anyone. See

http://www.cynergi.net/prod/exportsql/ for latest version. By Pedro Freire, support@cynergi.net. Note: Doesn't work with Access2!
- access_to_mysql.txt Paste this function into an Access module of a database which has the tables you want to export. See also `exportsql`. By Brian Andrews. Note: Doesn't work with Access2!
- importsql.txt A script that does the exact reverse of `exportsql.txt`. That is, it imports data from ***MySQL*** into an Access database via ODBC. This is very handy when combined with exportSQL, since it lets you use Access for all DB design and administration, and synchronize with your actual ***MySQL*** server either way. Free of charge. See http://www.netdive.com/freebies/importsql/ for any updates. Created by Laurent Bossavit of NetDIVE. ***Note:*** Doesn't work with Access2!
- /msql2mysqlWrapper 1.0 A C wrapper from `mSQL` to ***MySQL***. By alfred@sb.net
- sqlconv.pl A simple script that can be used to copy fields from one ***MySQL*** table to another in bulk. Basically, you can run `mysqldump` and pipe it to the `sqlconv.pl` script and the script will parse through the `mysqldump` output and will rearrange the fields so they can be inserted into a new table. An example is when you want to create a new table for a different site you are working on, but the table is just a bit different (ie − fields in different order, etc.). By Steve Shreeve.

B.6 Utilisateurs de MySQL avec d'autres produits

- emacs−sql−mode.tar.gz Raw port of a SQL mode for XEmacs. Supports completion. Original by Peter D. Pezaris pez@atlantic2.sbi.com and partial ***MySQL*** port by David Axmark.
- MyAccess.mda MyAccess is an AddIn for Access 97 and handles a lot of maintanance work for ***MySQL*** databases. MyAccess−readme. By Hubertus Hiden.
- radius−0.3.tar.gz Patches for `radiusd` to make it support ***MySQL***. By Wim Bonis,

B.7 Outils pratiques

- mysql_watchdog.pl Monitor the ***MySQL*** daemon for possible lockups. By Yermo Lamers,
- mysqltop.tar.gz Sends a query in a fixed time interval to the server and shows the resulting table. By Thomas Wana.
- http://www.mysql.com/Contrib/mysql_structure_dumper.tar.gz Prints out the structure of the all tables in a database. By Thomas Wana.
- structure_dumper.tgz Prints the structure of every table in a database. By Thomas Wana.
- mysqlsync−1.0−alpha.tar.gz. A perl script to keep remote copies of a ***MySQL*** database in sync with a central master copy. By Mark Jeftovic. markjr@easydns.com
- MySQLTutor. MySQLTutor. A tutor of ***MySQL*** for beginners
- MySQLDB.zip A COM library for ***MySQL*** by Alok Singh.
- MySQLDB−readme.html
- mysql_replicate.pl Perl program that handles replication. By mjs@atnet.net.au
- DBIx−TextIndex−0.02.tar.gz Perl program that uses reverse indexing to handle text searching. By Daniel Koch.

B.8 Divers

- findres.pl Find reserved words in tables. By Nem W Schlecht.
- handicap.tar.gz Performance handicapping system for yachts. Uses PHP. By
- hylalog−1.0.tar.gz Store `hylafax` outgoing faxes in a ***MySQL*** database. By Sinisa Milivojevic, sinisa@coresinc.com.
- mrtg−mysql−1.0.tar.gz ***MySQL*** status plotting with MRTG, by Luuk de Boer, luuk@wxs.nl.
- wuftpd−2.4.2.18−mysql_support.2.tar.gz Patches to add logging to ***MySQL*** for WU−ftpd. By Zeev Suraski,
- Old−Versions Previous versions of things found here that you probably won't be interested in.

C Contributions à MySQL

Les contributeurs à la distribution ***MySQL*** sont listés ci dessous, dans un ordre aléatoire :

Michael (Monty) Widenius
A écrit les parties suivantes de ***MySQL***:
- ◊ Tout le code principal de `mysqld`.
- ◊ De nouvelles fonctions pour la bibliothèques de chaînes.
- ◊ La plus grande partie de la bibliothèque `mysys`.
- ◊ Les bibliothèques `NISAM` et `MyISAM` (fichier à index B−tree et compression d'index, multi format de ligne).
- ◊ La librairie `heap`. Un système de mémoire en table, avec notre hashage supérieur. Utilisé depuis 1981, et publié en 1984.
- ◊ Le programme `replace` (regardez le, c'est grandiose).
- ◊ ***MyODBC***, le pilote ODBC pour Windows95.
- ◊ Correction de bugs dans MIT−pthreads pour faire fonctionner ***MySQL***. Et aussi Unireg, un utilitaire très puissant.
- ◊ Portage d'outils de `mSQL` comme `msqlperl`, `DBD/DBI` et `DB2mysql`.
- ◊ La plus part des testeurs mortels ("crash−me") et les benchmarks ***MySQL***.

David Axmark
- ◊ Coordinateur et rédacteur principal du ***Manuel de référence***, y compris les améliorations de `texi2html`. Mais aussi la mise à jour automatique du manuel sur le site.
- ◊ Support d'autoconf, automake et `libtool`.
- ◊ Les licences
- ◊ Une bonne partie de tous les fichiers textes.(Actuellement, seulement le fichier `` `README' `` reste de cette époque. Le reste est desormais dans le manuel).
- ◊ Notre Mailmaster.
- ◊ Tests de nombreuses nouveautés.
- ◊ Notre avocat maison et ``gratuit".
- ◊ Modérateur de la liste de diffusion (qui n'a jamais eu le temps de le faire proprement...)
- ◊ La portabilité originale du code (plus de 10 ans d'age, aujourd'hui..). Aujourd'hui, il reste encore des parties de `mysys`.
- ◊ Quelqu'un que Monty pouvait appeler au milieu de la nuit, juste pour lui parler de nouvelles fonctionnalités.

Paul DuBois
A permis l'achévement d'un manuel de référence correct et compréhensible.

Gianmassimo Vigazzola qwerg@mbox.vol.it or qwerg@tin.it
Le portage initial sur Win32/NT.

Kim Aldale
Réécriture des essais en anglais de Monty et David, en anglais.

Allan Larsson (le BOSS de TcX)
Pour tout le temps qu'il a laissé à Monty sur cet outil ``peut être utile" (***MySQL***). Utilisateur dévoué (et detecteur de bug) d'Unireg & ***MySQL***.

Per Eric Olsson
Pour ses critiques plus ou moins constructives, et ses tests réels sur le format dynamique de lignes.

Irena Pancirov irena@mail.yacc.it
Le portage de Win32 sous Borland compiler.

David J. Hughes
Pour ses efforts à faire une base de données SQL shareware . A TcX, nous avons commencé avec `mSQL`, mais nous ne l'avons pas trouvé satisfaisant pour nos objectifs, ce qui nous a poussé à écrire notre propre interface SQL pour Unireg. `mysqladmin` et `mysql` ont été largement influencé par leur alter ego `mSQL`. Nous avons mis un point d'honneur à faire de la syntaxe de ***MySQL*** un surensemble de celle de `mSQL`. De nombreux concepts d'APIs ont été emprunté à `mSQL` pour rendre plus facile le portage des programmes de `mSQL` à ***MySQL***. ***MySQL*** ne contient aucun code de `mSQL`. Les deux distributions (`` `client/insert_test.c' `` et `` `client/select_test.c' ``) sont basé sur les distributions (non−copyrighté) de la distribution `mSQL` mais ont été modifiés pour montrer les évolutions nécessaires entre `mSQL` et ***MySQL***. (`mSQL` est reservé par David J. Hughes.)

Fred Fish
Pour son excellente librairie de test C et sa librairie de tracage. Monty y a fait quelques améliorations mineurs (vitesse et options).

Richard A. O'Keefe
Pour sa librairie de chaîne du domaine public.

Henry Spencer
Pour sa bibliothèque regex, utilisée dans `WHERE colonne REGEXP expression_régulière`.

Free Software Foundation
Qui a fourni un excellent compilateur (`gcc`), la bibliothèque `libc` (dont nous avons emprunté `` `strto.c' `` pour faire fonctionner nos sources sur Linux) et la bibliothèque `readline` (pour le client `mysql`).

Free Software Foundation & The XEmacs development team
Pour un éditeur vraiment génial, et un environnement utilisé par tout le monde à TcX/detron.

Igor Romanenko igor@frog.kiev.ua
`mysqldump` (feu `msqldump`, mais porté est amélioré par Monty).

Tim Bunce, Alligator Descartes
Pour le code de l'interface Perl `DBD`

Andreas Koenig a.koenig@mind.de
Pour l'interface Perl de ***MySQL***.

Eugene Chan *eugene@acenet.com.sg*
Pour avoir porté PHP sur ***MySQL***.

Michael J. Miller Jr. *mke@terrapin.turbolift.com*
Pour le premier manuel ***MySQL***. Et pour les corrections orthographiques dans la FAQ (qui s'est mué en manuel ***MySQL*** il y a bien longtemps).

Giovanni Maruzzelli *maruzz@matrice.it*
Pour le portage de iODBC (Unix ODBC).

Chris Provenzano
Porage des pthreads. D'après le copyright: This product includes software developed by Chris Provenzano, the University of California, Berkeley, and contributors. Nous utilisons actuellement la version 1_60_beta6 patché par Monty (voir `mit-pthreads/Changes-mysql').

Xavier Leroy *Xavier.Leroy@inria.fr*
L'auteur de LinuxThreads (utilisé par ***MySQL*** sur Linux).

Zarko Mocnik *zarko.mocnik@dem.si*
Tri pour la langue Slovene et le module `cset.tar.gz' qui facilite l'ajout de nouveau jeu de caractères.

"TAMITO" *tommy@valley.ne.jp*
La macro _MB et les jeu de caractères ujis et sjis .

Yves Carlier *Yves.Carlier@rug.ac.be*
`mysqlaccess`, un programm qui montrer les droits d'accès d'un utilisateur.

Rhys Jones *rhys@wales.com* *(And GWE Technologies Limited)*
Pour le pilote JDBC, un module qui extrait les données de ***MySQL*** avec un client Java.

Dr Xiaokun Kelvin ZHU *X.Zhu@brad.ac.uk*
Pour les développement du pilote JDBC et d'autres outils liants ***MySQL*** et Java.

James Cooper *pixel@organic.com*
Pour avoir monté une liste de diffusion avec un outil de recherche, sur son site.

Rick Mehalick *Rick_Mehalick@i−o.com*
Pour `xmysql`, un client X graphique de ***MySQL***.

Doug Sisk *sisk@wix.com*
Pour avoir fourni les packages RPM de RedHat Linux pour ***MySQL***.

Diemand Alexander V. *axeld@vial.ethz.ch*
Pour avoir fourni les packages RPM de Linux/Alpha pour ***MySQL***.

Antoni Pamies Olive *toni@readysoft.es*
Pour avoir fourni les packages RPM d'Intel et Sparc pour ***MySQL***.

Jay Bloodworth *jay@pathways.sde.state.sc.us*
Pour avoir fourni les versions RPM de ***MySQL*** 3.21.

Jochen Wiedmann *wiedmann@neckar−alb.de*
Pour le support du module Perl `DBD::mysql`.

Therrien Gilbert *gilbert@ican.net*, *Jean−Marc Pouyot* *jmp@scalaire.fr*
Pour les messages d'erreur en Francais.

Petr snajdr, *snajdr@pvt.net*
Pour les messages d'erreur en Tchèque.

Jaroslaw Lewandowski *jotel@itnet.com.pl*
Pour les messages d'erreur en Polonais.

Miguel Angel Fernandez Roiz
Pour les messages d'erreur en Espagnol.

Roy−Magne Mo *rmo@www.hivolda.no*
Pour les messages d'erreur en Norvégien, et les tests des versions 3.21.#.

Timur I. Bakeyev *root@timur.tatarstan.ru*
Pour les messages d'erreur en Russe.

brenno@dewinter.com *&Filippo Grassilli* *phil@hyppo.com*
Pour les messages d'erreur en Italien.

Dirk Munzinger *dirk@trinity.saar.de*
Pour les messages d'erreur en Allemand.

Billik Stefan *billik@sun.uniag.sk*
Pour les messages d'erreur en Slovaque.

David Sacerdote *davids@secnet.com*
Pour ses idées sur la vérification sécurisante des nom de DNS.

Wei−Jou Chen *jou@nematic.ieo.nctu.edu.tw*
Pour le support des caractères Chinois (Chinese(BIG5).

Zeev Suraski *bourbon@netvision.net.il*
`FROM_UNIXTIME()`, format d'heure, fonctions `ENCRYPT()` et pour les conseils sur `bison`. Membre actif de la mailing liste.

Luuk de Boer *luuk@wxs.nl*
Portage (et amélioration) de la suite de test pour `DBI`/`DBD`. A été d'une grande aide avec `crash-me` et les benchmarks. Nouvelles fonctions de dates. Le script mysql_setpermissions.

Jay Flaherty *fty@utk.edu*
Grandes parties de la section du manuel pour `DBI`/`DBD`.

Paul Southworth *pauls@etext.org*, *Ray Loyzaga* *yar@cs.su.oz.au*
Relecteuf attentif du manuel de référence.

Alexis Mikhailov *root@medinf.chuvashia.su*
Fonction définies par les utilisateurs (UDFs); `CREATE FUNCTION` et `DROP FUNCTION`.

Andreas F. Bobak bobak@relog.ch
: L'extension `AGGREGATE` des fonctions UDF.

Ross Wakelin R.Wakelin@march.co.uk
: Aide avec InstallShield pour MySQL−Win32.

Jethro Wright III jetman@li.net
: La librairie `` `libmysql.dll' ``.

James Pereria jpereira@iafrica.com
: Mysqlmanager, un outil GUI Win32 pour administrer ***MySQL***.

Curt Sampson cjs@portal.ca
: Portage de MIT−pthreads vers NetBSD/Alpha et NetBSD 1.3/i386.

Sinisa Milivojevic sinisa@coresinc.com
: Compression (avec `zlib`) du protocol client/server. Hashage parfait pour l'analyseur lexical.

Antony T. Curtis antony.curtis@olcs.net
: Portage de ***MySQL*** vers OS/2.

Martin Ramsch m.ramsch@computer.org
: Exemples de tutorial pour ***MySQL***.

D'autres contributeurs, chercheur de bug, testeurs : James H. Thompson, Maurizio Menghini, Wojciech Tryc, Luca Berra, Zarko Mocnik, Wim Bonis, Elmar Haneke, jehamby@lightside, psmith@BayNetworks.COM, Mike Simons, Jaakko Hyv@"atti.

Et un grand nombre de retour de bug/patch des gars de la liste de diffusion.

Et un grand merci à ceux qui nous on aidé à répondre aux questions sur la liste `mysql@lists.mysql.com`:

Daniel Koch dkoch@amcity.com
: Installation Irix.

Luuk de Boer luuk@wxs.nl
: Questions de benchmark.

Tim Sailer tps@users.buoy.com
: Questions sur `DBD−mysql`.

Boyd Lynn Gerber gerberb@zenez.com
: Questions sur SCO.

Richard Mehalick RM186061@shellus.com
: Questions sur `xmysql` et problèmes basiques d'installation.

Zeev Suraski bourbon@netvision.net.il
: Questions sur la configuration du module Apache, Questions sur PHP Questions sur la syntaxe SQL Questions générales.

Francesc Guasch frankie@citel.upc.es
: Questions générales.

Jonathan J Smith jsmith@wtp.net
: Questions sur les OS spécifiques (Linux−like), les syntaxes SQL, et d'autres trucs encore.

David Sklar sklar@student.net
: Utilisation de ***MySQL*** à partir de PHP et Perl.

Alistair MacDonald A.MacDonald@uel.ac.uk
: Pas spécifique, mais peut gérer les problèmes sur Linux et parfois HP−UX. Poussera l'utilisateur à lancer `mysqlbug`.

John Lyon jlyon@imag.net
: Questions sur l'installation de ***MySQL*** sur les systèmes Linux, soit avec les fichiers `` `.rpm' `` , ou à partir de la source.

Lorvid Ltd. lorvid@WOLFENET.com
: Tout ce qui a trait aux licence, paiement, support.

Patrick Sherrill patrick@coconet.com
: Questions sur l'interface ODBC et VisualC++.

Randy Harmon rjharmon@uptimecomputers.com
: `DBD`, Linux, et des questiosn de syntaxe SQL.

D Historique des versions de MySQL

Notez bien que nous tachons de mettre à jour ce manuel en même temps que nous implémentons ***MySQL***. Si vous trouver une version ici, qui n'est pas listée sur la page ***MySQL*** , c'est que c'est une version qui n'a pas encore été publiée.

D.1 Modifications de la version 3.23.x (Released as alpha)

The major difference between release 3.23 and releases 3.22 and 3.21 is that 3.23 contains a new ISAM library (MyISAM), which is more tuned for SQL than the old ISAM was.

The 3.23 release is under development, and things will be added at a fast pace to it. For the moment we recommend this version only for users that desperately need a new feature that is found only in this release (like big file support and machine−independent tables). (Note that all new functionality in MySQL 3.23 is extensively tested, but as this release involves much new code, it's difficult to test everything). This version should start to stabilise as soon as we get subselects included in it.

D.1.1 Modifications de la version 3.23.3

- `ASC` is now again default for `ORDER BY`.
- Added `LIMIT` to `UPDATE`.
- New client function: `mysql_change_user`.
- Added character set to `SHOW VARIABLES`
- Added support of `−−[whitespace]` comments.
- Allow `INSERT into table_name VALUES ()'`
- Changed `SUBSTRING(text FROM pos)` to conform to ANSI SQL. (Before this construct returned the rightmost 'pos' characters).
- `SUM(..)` with `GROUP BY` returned 0 on some system.
- Changed output for `SHOW TABLE STATUS`.
- Added `DELAY_KEY_WRITE` option to `CREATE TABLE`.
- Allow `AUTO_INCREMENT` on any key part.
- Fixed problem with `YEAR(NOW())` and `YEAR(CURDATE())`.
- Added `CASE` construct.
- New function `COALESCE()`.

D.1.2 Modifications de la version 3.23.2

- Fixed range optimizer bug: `SELECT * FROM table_name WHERE key_part1 >= const AND (key_part2 = const OR key_part2 = const)`. The bug was that some rows could be duplicated in the result.
- Running `myisamchk` without `−a` updated the index distribution wrong.
- `SET SQL_LOW_PRIORITY_UPDATES=1` gave parse error before.
- You can now update indexes columns that are used in the `WHERE` clause. `UPDATE nom_table SET KEY=KEY+1 WHERE KEY > 100`
- Date handling should now be a bit faster.
- Added handling of fuzzy dates (dates where day or month is 0): (Like: 1999−01−00)
- Fixed optimization of `SELECT ... WHERE key_part1=const1 AND key_part_2=const2 AND key_part1=const4 AND key_part2=const4` ; Indextype should be `range` instead of `ref`.
- Fixed `egcs` 1.1.2 optimizer bug (when using `BLOB`s) on Linux Alpha.
- Fixed problem with `LOCK TABLES` combined with `DELETE FROM table`.
- MyISAM tables now allow keys on `NULL` and `BLOB/TEXT` columns.
- The following join is now much faster: `SELECT ... FROM t1 LEFT JOIN t2 ON ... WHERE t2.not_null_column IS NULL`.
- `ORDER BY` and `GROUP BY` can be done on functions.
- Changed handling of 'const_item' to allow handling of `ORDER BY RAND()`.
- Indexes are now used for `WHERE key_column = function`.
- Indexes are now used for `WHERE key_column = column_name` even if the columns are not identically packed.
- Indexes are now used for `WHERE column_name IS NULL`.
- Changed heap tables to be stored in low_byte_first order (to make it easy to convert to MyISAM tables)
- Automatic change of HEAP temporary tables to MyISAM tables in case of 'table is full' errors.
- Added option `−−init−file=file_name` to `mysqld`.

- `COUNT(DISTINCT value,[value,...])`
- `CREATE TEMPORARY TABLE` now creates a temporary table, in its own namespace, that is automatically deleted if connection is dropped.
- New keywords (required for `CASE`: `CASE, THEN, WHEN, ELSE and END`.
- New functions `EXPORT_SET()` and `MD5()`.
- Support for the GB2312 Chinese character set.

D.1.3 Modifications de la version 3.23.1

- Fixed some compilation problems.

D.1.4 Modifications de la version 3.23.0

A new table handler library (`MyISAM`) with a lot of new features. 10.18 Types de tables MySQL.

- You can create in−memory `HEAP` tables which are extremely fast for lookups.
- Support for big files (63 bit) on OSes that support big files.
- New function `LOAD_FILE(filename)` to get the contents of a file as a string value.
- New operator `<=>` which will act as `=` but will return TRUE if both arguments are `NULL`. This is useful for comparing changes between tables.
- Added the ODBC 3.0 `EXTRACT(interval FROM datetime)` function.
- Columns defined as `FLOAT(4)` or `FLOAT(8)` are not rounded on storage and may be in scientific notation (1.0 E+10) when retrieved.
- `REPLACE` is now faster than before.
- Changed `LIKE` character comparison to behave as `=`; This means that `'e' LIKE 'é'` is now true.
- `SHOW TABLE STATUS` returns a lot of information about the tables.
- Added `LIKE` to the `SHOW STATUS` command.
- Added privilege column to `SHOW COLUMNS`.
- Added columns `packed` and `comment` to `SHOW INDEX`.
- Added comments to tables (with `CREATE TABLE ... COMMENT "xxx"`).
- Added `UNIQUE`, as in `CREATE TABLE table_name (col int not null UNIQUE)`
- New create syntax: `CREATE TABLE SELECT ....`
- New create syntax: `CREATE TABLE IF NOT EXISTS ...`
- Allow creation of `CHAR(0)` columns.
- `DATE_FORMAT()` now requires `%` before any format character.
- `DELAYED` is now a reserved word (sorry about that :( ).
- An example procedure is added: `analyse`, file: `` `sql_analyse.c' ``. This will describe the data in your query. Try the following:
  `SELECT ... FROM ... WHERE ... PROCEDURE ANALYSE([max elements,[max memory]])`

  This procedure is extremely useful when you want to check the data in your table!

- `BINARY` cast to force a string to be compared case sensitively.
- Added option `--skip-show-databases` to `mysqld`.
- Check if a row has changed in an `UPDATE` now also works with `BLOB`/`TEXT` columns.
- Added the `INNER` join syntax. ***NOTE***: This made `INNER` an reserved word!
- Added support for netmasks to the hostname in the ***MySQL*** tables. You can specify a netmask using the `IP/NETMASK` syntax.
- If you compare a `NOT NULL DATE/DATETIME` column with `IS NULL`, this is changed to a compare against `0` to satisfy some ODBC applications. (By shreeve@uci.edu).
- `NULL IN (...)` now returns `NULL` instead of `0`. This will ensure that `null_column NOT IN (...)` doesn't match `NULL` values.
- Fix storage of floating point values in `TIME` columns.
- Changed parsing of `TIME` strings to be more strict. Now the fractional second part is detected (and currently skipped). The following formats are supported:
  `[[DAYS] [H]H:]MM:]SS[.fraction]`
  `[[[[[H]H]H]H]MM]SS[.fraction]`
- Detect (and ignore) second fraction part from `DATETIME`.
- Added the `LOW_PRIORITY` attribute to `LOAD DATA INFILE`.
- The default index name now uses the same case as the used column name.
- Changed default number of connections to 100.
- Use bigger buffers when using `LOAD DATA INFILE`.
- `DECIMAL(x,y)` now works according to ANSI SQL.
- Added aggregate UDF functions. Thanks to Andreas F. Bobak
- `LAST_INSERT_ID()` is now updated for `INSERT INTO ... SELECT`.
- Some small changes to the join table optimizer to make some joins faster.
- `SELECT DISTINCT` is much faster; It uses the new `UNIQUE` functionality in `MyISAM`. One difference compared to ***MySQL*** 3.22 is that the output of `DISTINCT` is not sorted anymore.
- All C client API macros are now functions to make shared libraries more reliable. Because of this, you can no longer call `mysql_num_fields()` on a `MYSQL` object, you must use `mysql_field_count()` instead.

- Added use of `LIBEWRAP`; Patch by Henning P . Schmiedehausen.
- Don't allow `AUTO_INCREMENT` for other than numerical columns.
- Using `AUTO_INCREMENT` will now automatically make the column `NOT NULL`.
- Show `NULL` as the default value for AUTO_INCREMENT columns.
- Added `SQL_BIG_RESULT`; `SQL_SMALL_RESULT` is now default.
- Added a shared library RPM. This enchancement was contributed by David Fox (dsfox@cogsci.ucsd.edu).
- Added a `--enable-large-files/--disable-large-files` switch to `configure`. See `` `configure.in' `` for some systems where this is automatically turned off because of broken implementations.
- Upgraded `readline` to 4.0.
- New `CREATE TABLE` options: `PACK_KEYS` and `CHECKSUM`.
- Added `mysqld` option `--default-table-type`.

D.2 Modifications de la version 3.22.x

The 3.22 version has faster and safer connect code and a lot of new nice enhancements. The reason for not including these changes in the 3.21 version is mainly that we are trying to avoid big changes to 3.21 to keep it as stable as possible. As there aren't really any MAJOR changes, upgrading to 3.22 should be very easy and painless. 4.16.2 Mise à jour de a 3.21 vers 3.22.

3.22 should also be used with the new `DBD-mysql` (1.20xx) driver that can use the new connect protocol!

D.2.1 Modifications de la version 3.22.26

- Fixed core dump with empty `BLOB/TEXT` column to `REVERSE()`.
- Extended `/*! */` with version numbers.
- Changed `SUBSTRING(text FROM pos)` to conform to ANSI SQL. (Before this construct returned the rightmost 'pos' characters).
- Fixed problem with `LOCK TABLES` combined with `DELETE FROM table`
- Fixed problem that INSERT ... SELECT didn't use SQL_BIG_TABLES.
- `SET SQL_LOW_PRIORITY_UPDATES=#` didn't work.
- Password wasn't updated correctly if privileges didn't change on: `GRANT ... IDENTIFIED BY`
- Fixed range optimizer bug in `SELECT * FROM table_name WHERE key_part1 >= const AND (key_part2 = const OR key_part2 = const)`
- Fixed bug in compression key handling in NISAM.

D.2.2 Modifications de la version 3.22.25

- Fixed some small problems with the installation.

D.2.3 Modifications de la version 3.22.24

- `DATA` is not a reserved word anymore.
- Fixed optimizer bug with tables with only one row.
- Fixed bug when using `LOCK TABLES table_name READ; FLUSH TABLES;`
- Applied some patches for HP−UX.
- `isamchk` should now work on Win32.
- Changed `` `configure' `` to not use big file handling on Linux as this crashes some RedHat 6.0 systems

D.2.4 Modifications de la version 3.22.23

- Upgraded to use Autoconf 2.13, Automake 1.4 and `libtool` 1.3.2.
- Better support for SCO in `configure`.
- Added option `--defaults-file=###` to option file handling to force use of only one specific option file.
- Extended `CREATE` syntax to ignore `MySQL 3.23` keywords.
- Fixed deadlock problem when using `INSERT DELAYED` on a table locked with `LOCK TABLES`.
- Fixed deadlock problem when using `DROP TABLE` on a table that was locked by another thread.
- Add logging of `GRANT/REVOKE` commands in the update log.
- Fixed `isamchk` to detect a new error condition.
- Fixed bug in `NATURAL LEFT JOIN`.

D.2.5 Modifications de la version 3.22.22

- Fixed problem in the C API when you called `mysql_close()` directly after `mysql_init()`.
- Better client error message when you can't open socket.
- Fixed `delayed_insert_thread` counting when you couldn't create a new delayed_insert thread.
- Fixed bug in `CONCAT()` with many arguments.
- Added patches for DEC 3.2 and SCO.
- Fixed path−bug when installing ***MySQL*** as a service on NT.
- The ***MySQL***−win32 version is now compiled with VC++ 6.0 instead of with VC++ 5.0.
- New installation setup for ***MySQL−Win32***

D.2.6 Modifications de la version 3.22.21

- Fixed problem with `DELETE FROM TABLE` when table was locked by another thread.
- Fixed bug in `LEFT JOIN` involving empty tables.
- Changed the `mysql.db` column from `char(32)` to `char(60)`.
- `MODIFY` and `DELAYED` are not reserved words anymore.
- Fixed a bug when storing days in a `TIME` column.
- Fixed a problem with `Host '..' is not allowed to connect to this MySQL server` after one had inserted a new ***MySQL*** user with a `GRANT` command.
- Changed to use TCP_NODELAY also on Linux (Should give faster TCP/IP connections).

D.2.7 Modifications de la version 3.22.20

- Fixed `STD()` for big tables when result should be 0.
- The update log didn't have newlines on some operating systems.
- `INSERT DELAYED` had some garbage at end in the update log.

D.2.8 Modifications de la version 3.22.19

- Fixed bug in `mysql_install_db` (from 3.22.17).
- Changed default key cache size to 8M.
- Fixed problem with requêtesthat needed temporary tables with `BLOB` columns.

D.2.9 Modifications de la version 3.22.18

- Fixes a fatal problem in 3.22.17 on Linux; After `shutdown` all threads didn't die properly.
- Added option `−O flush−time=#` to `mysqld`. This is mostly useful on Win32 and tells how often ***MySQL*** should close all unused tables and flush all updated tables to disk.
- Fixed problem that a `VARCHAR` column compared with `CHAR` column didn't use keys efficiently.

D.2.10 Modifications de la version 3.22.17

- Fixed a core dump problem when using `−−log−update` and connecting without a default database.
- Fixed some `configure` and portability problems.
- Using `LEFT JOIN` on tables that had circular dependencies caused `mysqld` to hang forever.

D.2.11 Modifications de la version 3.22.16

- `mysqladmin processlist` could kill the server if a new user logged in.
- `DELETE FROM nom_table WHERE key_column=nom_colonne` didn't find any matching rows. Fixed.
- `DATE_ADD(column,...)` didn't work.
- `INSERT DELAYED` could deadlock with status 'upgrading lock'
- Extended `ENCRYPT()` to take longer salt strings than 2 characters.
- `longlong2str` is now much faster than before. For `Intel x86` platforms, this function is written in optimized assembler.
- Added the `MODIFY` keyword to `ALTER TABLE`.

D.2.12 Modifications de la version 3.22.15

- GRANT used with IDENTIFIED BY didn't take effect until privileges were flushed.
- Name change of some variables in SHOW STATUS.
- Fixed problem with ORDER BY with 'only index' optimzation when there were multiple key definitions for a used column.
- DATE and DATETIME columns are now up to 5 times faster than before.
- INSERT DELAYED can be used to let the client do other things while the server inserts rows into a table.
- LEFT JOIN USING (col1,col2) didn't work if one used it with tables from 2 different databases.
- LOAD DATA LOCAL INFILE didn't work in the Unix version because of a missing file.
- Fixed problems with VARCHAR/BLOB on very short rows (< 4 bytes); error 127 could occur when deleting rows.
- Updating BLOB/TEXT through formulas didn't work for short (< 256 char) strings.
- When you did a GRANT on a new host, mysqld could die on the first connect from this host.
- Fixed bug when one used ORDER BY on column name that was the same name as an alias.
- Added BENCHMARK(loop−count,expression) function to time expressions.

D.2.13 Modifications de la version 3.22.14

- Allow empty arguments to mysqld to make it easier to start from shell scripts.
- Setting a TIMESTAMP column to NULL didn't record the timestamp value in the update log.
- Fixed lock handler bug when one did INSERT INTO TABLE ... SELECT ... GROUP BY.
- Added a patch for localtime_r() on Win32 so that it will not crash anymore if your date is > 2039, but instead will return a time of all zero.
- Names for user−defined functions are no longer case sensitive.
- Added escape of ^Z (ASCII 26) to \Z as ^Z doesn't work with pipes on Win32.
- mysql_fix_privileges adds a new column to the mysql.func to support aggregate UDF functions in future ***MySQL*** releases.

D.2.14 Modifications de la version 3.22.13

- Saving NOW(), CURDATE() or CURTIME() directly in a column didn't work.
- SELECT COUNT(*) ... LEFT JOIN ... didn't work with no WHERE part.
- Updated `config.guess' to allow ***MySQL*** to configure on UnixWare 7.0.x.
- Changed the implementation of pthread_cond() on the Win32 version. get_lock() now correctly times out on Win32!

D.2.15 Modifications de la version 3.22.12

- Fixed problem when using DATE_ADD() and DATE_SUB() in a WHERE clause.
- You can now set the password for a user with the GRANT ... TO user IDENTIFIED BY 'password' syntax.
- Fixed bug in GRANT checking with SELECT on many tables.
- Added missing file mysql_fix_privilege_tables to the RPM distribution. This is not run by default since it relies on the client package.
- Added option SQL_SMALL_RESULT to SELECT to force use of fast temporary tables when you know that the result set will be small.
- Allow use of negative real numbers without a decimal point.
- Day number is now adjusted to max days in month if the resulting month after DATE_ADD/DATE_SUB() doesn't have enough days.
- Fix that GRANT compares columns in case−insensitive fashion.
- Fixed a bug in `sql_list.h' that made ALTER TABLE dump core in some contexts.
- The hostname in user@hostname can now include `.' and `−' without quotes in the context of the GRANT, REVOKE and SET PASSWORD FOR ... statements.
- Fix for isamchk for tables which need big temporary files.

D.2.16 Modifications de la version 3.22.11

- ***IMPORTANT***: You must run the mysql_fix_privilege_tables script when you upgrade to this version! This is needed because of the new GRANT system. If you don't do this, you will get Access denied when you try to use ALTER TABLE, CREATE INDEX or DROP INDEX.
- GRANT to allow/deny users table and column access.
- Changed USER() to return user@host
- Changed the syntax for how to set PASSWORD for another user.
- New command FLUSH STATUS that sets most status variables to zero.
- New status variables: aborted_threads, aborted_connects.
- New option variable: connection_timeout.
- Added support for Thai sorting (by Pruet Boonma
- Slovak and japanese error messages.
- Configuration and portability fixes.

- Added option `SET SQL_WARNINGS=1` to get a warning count also for simple inserts.
- ***MySQL*** now uses `SIGTERM` instead of `SIGQUIT` with shutdown to work better on FreeBSD.
- Added option `\G` (print vertically) to `mysql`.
- `SELECT HIGH_PRIORITY` ... killed `mysqld`.
- `IS NULL` on a `AUTO_INCREMENT` column in a `LEFT JOIN` didn't work as expected.
- New function `MAKE_SET()`.

D.2.17 Modifications de la version 3.22.10

- `mysql_install_db` no longer starts the ***MySQL*** server! You should start `mysqld` with `safe_mysqld` after installing it! The ***MySQL*** RPM will however start the server as before.
- Added `--bootstrap` option to `mysqld` and recoded `mysql_install_db` to use it. This will make it easier to install ***MySQL*** with RPMs.
- Changed `+`, `−` (sign and minus), `*`, `/`, `%`, `ABS()` and `MOD()` to be `BIGINT` aware (64−bit safe).
- Fixed a bug in `ALTER TABLE` that caused `mysqld` to crash.
- ***MySQL*** now always reports the conflicting key values when a duplicate key entry occurs. (Before this was only reported for `INSERT`).
- New syntax: `INSERT INTO nom_table SET nom_colonne=value,nom_colonne=value,...`
- Most errors in the `` `.err' `` log are now prefixed with a time stamp.
- Added option `MYSQL_INIT_COMMAND` to `mysql_options()` to make a query on connect or reconnect.
- Added option `MYSQL_READ_DEFAULT_FILE` and `MYSQL_READ_DEFAULT_GROUP` to `mysql_options()` to read the following parameters from the ***MySQL*** option files: port, socket, compress, password, pipe, timeout, user, init−command, host and database.
- Added `maybe_null` to the UDF structure.
- Added option `IGNORE` to `INSERT` statemants with many rows.
- Fixed some problems with sorting of the koi8 character sets; Users of koi8 ***MUST*** run `isamchk -rq` on each table that has an index on a `CHAR` or `VARCHAR` column.
- New script `mysql_setpermission`, by Luuk de Boer, allows one to easily create new users with permissions for specific databases.
- Allow use of hexadecimal strings (0x...) when specifying a constant string (like in the column separators with `LOAD DATA INFILE`).
- Ported to OS/2 (thanks to Antony T. Curtis antony.curtis@olcs.net).
- Added more variables to `SHOW STATUS` and changed format of output to be like `SHOW VARIABLES`.
- Added `extended−status` command to `mysqladmin` which will show the new status variables.

D.2.18 Modifications de la version 3.22.9

- `SET SQL_LOG_UPDATE=0` caused a lockup of the server.
- New SQL command: `FLUSH [ TABLES | HOSTS | LOGS | PRIVILEGES ] [, ...]`
- New SQL command: `KILL` thread_id
- Added casts and changed include files to make ***MySQL*** easier to compile on AIX and DEC OSF1 4.x
- Fixed conversion problem when using `ALTER TABLE` from a `INT` to a short `CHAR()` column.
- Added `SELECT HIGH_PRIORITY`; This will get a lock for the `SELECT` even if there is a thread waiting for another `SELECT` to get a `WRITE LOCK`.
- Moved wild_compare to string class to be able to use `LIKE` on `BLOB`/`TEXT` columns with `\0`.
- Added `ESCAPE` option to `LIKE`.
- Added a lot more output to `mysqladmin debug`.
- You can now start `mysqld` on Win32 with the `--flush` option. This will flush all tables to disk after each update. This makes things much safer on NT/Win98 but also ***MUCH*** slower.

D.2.19 Modifications de la version 3.22.8

- Czech character sets should now work much better. You must also install ftp://www.mysql.com/pub/mysql/Downloads/Patches/czech−3.22.8−patch. This patch should also be installed if you are using a character set with uses `my_strcoll()`! The patch should always be safe to install (for any system), but as this patch changes ISAM internals it's not yet in the default distribution.
- `DATE_ADD()` and `DATE_SUB()` didn't work with group functions.
- `mysql` will now also try to reconnect on `USE DATABASE` commands.
- Fix problem with `ORDER BY` and `LEFT JOIN` and `const` tables.
- Fixed problem with `ORDER BY` if the first `ORDER BY` column was a key and the rest of the `ORDER BY` columns wasn't part of the key.
- Fixed a big problem with `OPTIMIZE TABLE`.
- ***MySQL*** clients on NT will now by default first try to connect with named pipes and after this with TCP/IP.
- Fixed a problem with `DROP TABLE` and `mysqladmin shutdown` on Win32 (a fatal bug from 3.22.6).
- Fixed problems with `TIME columns` and negative strings.
- Added an extra thread signal loop on shutdown to avoid some error messages from the client.
- ***MySQL*** now uses the next available number as extension for the update log file.
- Added patches for UNIXWARE 7.

D.2.20 Modifications de la version 3.22.7

- Added `LIMIT` clause for the `DELETE` statement.
- You can now use the `/*! ... */` syntax to hide ***MySQL***−specific keywords when you write portable code. ***MySQL*** will parse the code inside the comments as if the surrounding `/*!` and `*/` comment characters didn't exist.
- `OPTIMIZE TABLE nom_table` can now be used to reclaim disk space after many deletes. Currently, this uses `ALTER TABLE` to re−generate the table, but in the future it will use an integrated `isamchk` for more speed.
- Upgraded `libtool` to get the configure more portable.
- Fixed slow `UPDATE` and `DELETE` operations when using `DATETIME` or `DATE` keys.
- Changed optimizer to make it better at deciding when to do a full join and when using keys.
- You can now use `mysqladmin proc` to display information about your own threads. Only users with the ***Process_priv*** privilege can get information about all threads.
- Added handling of formats `YYMMDD`, `YYYYMMDD`, `YYMMDDHHMMSS` for numbers when using `DATETIME` and `TIMESTAMP` types. (Formerly these formats only worked with strings.)
- Added connect option `CLIENT_IGNORE_SPACE` to allow use of spaces after function names and before `` `(' `` (Powerbuilder requires this). This will make all function names reserved words.
- Added the `--log−long−format` option to `mysqld` to enable timestamps and INSERT_ID's in the update log.
- Added `--where` option to `mysqldump` (patch by Jim Faucette).
- The lexical analyzer now uses ``perfect hashing'' for faster parsing of SQL statements.

D.2.21 Modifications de la version 3.22.6

- Faster `mysqldump`.
- For the `LOAD DATA INFILE` statement, you can now use the new `LOCAL` keyword to read the file from the client. `mysqlimport` will automatically use `LOCAL` when importing with the TCP/IP protocol.
- Fixed small optimize problem when updating keys.
- Changed makefiles to support shared libraries.
- ***MySQL***−NT can now use named pipes, which means that you can now use ***MySQL***−NT without having to install TCP/IP.

D.2.22 Modifications de la version 3.22.5

- All table lock handing is changed to avoid some very subtle deadlocks when using `DROP TABLE`, `ALTER TABLE`, `DELETE FROM TABLE` and `mysqladmin flush−tables` under heavy usage. Changed locking code to get better handling of locks of different types.
- Updated `DBI` to 1.00 and `DBD` to 1.2.0.
- Added a check that the error message file contains error messages suitable for the current version of `mysqld`. (To avoid errors if you accidentally try to use an old error message file.)
- All count structures in the client (`affected_rows()`, `insert_id()`,...) are now of type `BIGINT` to allow 64−bit values to be used. This required a minor change in the ***MySQL*** protocol which should affect only old clients when using tables with `AUTO_INCREMENT` values > 24M.
- The return type of `mysql_fetch_lengths()` has changed from `uint *` to `ulong *`. This may give a warning for old clients but should work on most machines.
- Change `mysys` and `dbug` libraries to allocate all thread variables in one struct. This makes it easier to make a threaded `` `libmysql.dll' `` library.
- Use the result from `gethostname()` (instead of `uname()`) when constructing `` `.pid' `` file names.
- New better compressed server/client protocol.
- `COUNT()`, `STD()` and `AVG()` are extended to handle more than 4G rows.
- You can now store values in the range `−838:59:59` <= x <= `838:59:59` in a `TIME` column.
- ***WARNING: INCOMPATIBLE CHANGE!!*** If you set a `TIME` column to too short a value, ***MySQL*** now assumes the value is given as: `[[[D ]HH:]MM:]SS` instead of `HH[:MM[:SS]]`.
- `TIME_TO_SEC()` and `SEC_TO_TIME()` can now handle negative times and hours up to 32767.
- Added new option `SET OPTION SQL_LOG_UPDATE={0|1}` to allow users with the ***process*** privilege to bypass the update log. (Modified patch from Sergey A Mukhin violet@rosnet.net.)
- Fixed fatal bug in `LPAD()`.
- Initialize line buffer in `` `mysql.cc' `` to make `BLOB` reading from pipes safer.
- Added `−O max_connect_errors=#` option to `mysqld`. Connect errors are now reset for each correct connection.
- Increased the default value of `max_allowed_packet` to `1M` in `mysqld`.
- Added `--low−priority−updates` option to `mysqld`, to give `UPDATE` operations lower priority than retrievals. You can now use `{INSERT | REPLACE | UPDATE | DELETE} LOW_PRIORITY ...` You can also use `SET OPTION LOW_PRIORITY_UPDATES={0|1}` to change the priority for one thread. One side effect is that `LOW_PRIORITY` is now a reserved word. :(
- Add support for `INSERT INTO table ... VALUES(...),(...),(...)`, to allow inserting multiple rows with a single statement.
- `INSERT INTO nom_table` is now also cached when used with `LOCK TABLES`. (Previously only `INSERT ... SELECT` and `LOAD DATA INFILE` were cached.)
- Allow `GROUP BY` functions with `HAVING`:

```
mysql62; SELECT col FROM table GROUP BY col HAVING COUNT(*)62;0;
```

- `mysqld` will now ignore trailing `` `;' `` characters in queries. This is to make it easier to migrate from some other SQL servers that require the trailing `` `;' ``.
- Fix for corrupted fixed−format output generated by `SELECT INTO OUTFILE`.
- ***WARNING: INCOMPATIBLE CHANGE!!*** Added Oracle `GREATEST()` and `LEAST()` functions. You must now use these instead of the `MAX()` and `MIN()` functions to get the largest/smallest value from a list of values. These can now handle `REAL`, `BIGINT` and string (`CHAR` or `VARCHAR`) values.
- ***WARNING: INCOMPATIBLE CHANGE!!*** `DAYOFWEEK()` had offset 0 for Sunday. Changed the offset to 1.
- Give an error for requêtesthat mix `GROUP BY` columns and fields when there is no `GROUP BY` specification.
- Added `--vertical` option to `mysql`, for printing results in vertical mode.
- Index−only optimization; some requêtesare now resolved using only indexes. Until ***MySQL*** 4.0, this works only for numeric columns. ***MySQL*** indexes.
- Lots of new benchmarks.
- A new C API chapter and lots of other improvements in the manual.

D.2.23 Modifications de la version 3.22.4

- Added `--tmpdir` option to `mysqld`, for specifying the location of the temporary file directory.
- ***MySQL*** now automatically changes a query from an ODBC client:

```
SELECT ... FROM table WHERE auto_increment_column IS NULL
```

to:

```
SELECT ... FROM table WHERE auto_increment_column == LAST_INSERT_ID()
```

This allows some ODBC programs (Delphi, Access) to retrieve the newly inserted row to fetch the `AUTO_INCREMENT` id.

- `DROP TABLE` now waits for all users to free a table before deleting it.
- Fixed small memory leak in the new connect protocol.
- New functions `BIN()`, `HEX()` and `CONV()` for converting between different number bases.
- Added function `SUBSTRING()` with 2 arguments.
- If you created a table with a record length smaller than 5, you couldn't delete rows from the table.
- Added optimization to remove const reference tables from `ORDER BY` and `GROUP BY`.
- `mysqld` now automatically disables system locking on Linux and Win32, and for systems that use MIT−pthreads. You can force the use of locking with the `--enable-locking` option.
- Added `--console` option to `mysqld`, to force a console window (for error messages) when using Win32.
- Fixed table locks for Win32.
- Allow `` `$' `` in identifiers.
- Changed name of user−specific configuration file from `` `my.cnf' `` to `` `.my.cnf' `` (Unix only).
- Added `DATE_ADD()` and `DATE_SUB()` functions.

D.2.24 Modifications de la version 3.22.3

- Fixed a lock problem (bug in ***MySQL*** 3.22.1) when closing temporary tables.
- Added missing `mysql_ping()` to the client library.
- Added `--compress` option to all ***MySQL*** clients.
- Changed `byte` to `char` in `` `mysql.h' `` and `` `mysql_com.h' ``.

D.2.25 Modifications de la version 3.22.2

- Searching on multiple constant keys that matched more than 30% of the rows didn't always use the best possible key.
- New functions <<, >>, `RPAD()` and `LPAD()`.
- You can now save default options (like passwords) in a configuration file (`` `my.cnf' ``).
- Lots of small changes to get `ORDER BY` to work when no records are found when using fields that are not in `GROUP BY` (***MySQL*** extension).
- Added `--chroot` option to `mysqld`, to start `mysqld` in a chroot environment (by Nikki Chumakov nikkic@cityline.ru).
- Trailing spaces are now ignored when comparing case−sensitive strings; this should fix some problems with ODBC and flag 512!
- Fixed a core−dump bug in the range optimizer.
- Added `--one-thread` option to `mysqld`, for debugging with LinuxThreads (or `glibc`). (This replaces the `-T32` flag)
- Added `DROP TABLE IF EXISTS` to prevent an error from occurring if the table doesn't exist.
- `IF` and `EXISTS` are now reserved words (they would have to be sooner or later).
- Added lots of new options to `mysqldump`.
- Server error messages are now in `` `mysqld_error.h' ``.
- The server/client protocol now supports compression.
- All bug fixes from ***MySQL*** 3.21.32.

D.2.26 Modifications de la version 3.22.1

- Added new C API function `mysql_ping()`.
- Added new API functions `mysql_init()` and `mysql_options()`. You now MUST call `mysql_init()` before you call `mysql_real_connect()`. You don't have to call `mysql_init()` if you only use `mysql_connect()`.
- Added `mysql_options(...,MYSQL_OPT_CONNECT_TIMEOUT,...)` so you can set a timeout for connecting to a server.
- Added `−−timeout` option to `mysqladmin`, as a test of `mysql_options()`.
- Added `AFTER column` and `FIRST` options to `ALTER TABLE ... ADD columns`. This makes it possible to add a new column at some specific location within a row in an existing table.
- `WEEK()` now takes an optional argument to allow handling of weeks when the week starts on Monday (some European countries). By default, `WEEK()` assumes the week starts on Sunday.
- `TIME` columns weren't stored properly (bug in ***MySQL*** 3.22.0).
- `UPDATE` now returns information about how many rows were matched and updated, and how many ``warnings'' occurred when doing the update.
- Fixed incorrect result from `FORMAT(−100,2)`.
- `ENUM` and `SET` columns were compared in binary (case−sensitive) fashion; changed to be case insensitive.

D.2.27 Modifications de la version 3.22.0

- New (backward compatible) connect protocol that allows you to specify the database to use when connecting, to get much faster connections to a specific database. The `mysql_real_connect()` call is changed to:

```
mysql_real_connect(MYSQL *mysql, const char *host, const char *user,
                   const char *passwd, const char *db, uint port,
                   const char *unix_socket, uint client_flag)
```

- Each connection is handled by its own thread, rather than by the master `accept()` thread. This fixes permanently the telnet bug that was a topic on the mail list some time ago.
- All TCP/IP connections are now checked with backward resolution of the hostname to get better security. `mysqld` now has a local hostname resolver cache so connections should actually be faster than before, even with this feature.
- A site automatically will be blocked from future connections if someone repeatedly connects with an ``improper header'' (like when one uses telnet).
- You can now refer to tables in different databases with references of the form `nom_table@nom_base_de_donnees` or `nom_base_de_donnees.nom_table`. This makes it possible to give a user read access to some tables and write access to others simply by keeping them in different databases!
- Added `−−user` option to `mysqld`, to allow it to run as another Unix user (if it is started as the Unix `root` user).
- Added caching of users and access rights (for faster access rights checking)
- Normal users (not anonymous ones) can change their password with `mysqladmin password 'new_password'`. This uses encrypted passwords that are not logged in the normal ***MySQL*** log!
- All important string functions are now coded in assembler for x86 Linux machines. This gives a speedup of 10% in many cases.
- For tables that have many columns, the column names are now hashed for much faster column name lookup (this will speed up some benchmark tests a lot!)
- Some benchmarks are changed to get better individual timing. (Some loops were so short that a specific test took < 2 seconds. The loops have been changed to take about 20 seconds to make it easier to compare different databases. A test that took 1−2 seconds before now takes 11−24 seconds, which is much better)
- Re−arranged `SELECT` code to handle some very specific queries involving group functions (like `COUNT(*)`) without a `GROUP BY` but with `HAVING`. The following now works:

```
mysql62; SELECT count(*) as C FROM table HAVING C 62; 1;
```

- Changed the protocol for field functions to be faster and avoid some calls to `malloc()`.
- Added `−T32` option to `mysqld`, for running all requêtesunder the main thread. This makes it possible to debug `mysqld` under Linux with `gdb`!
- Added optimization of `not_null_column IS NULL` (needed for some Access queries).
- Allow `STRAIGHT_JOIN` to be used between two tables to force the optimizer to join them in a specific order.
- String functions now return `VARCHAR` rather than `CHAR` and the column type is now `VARCHAR` for fields saved as `VARCHAR`. This should make the ***MyODBC*** driver better, but may break some old ***MySQL*** clients that don't handle `FIELD_TYPE_VARCHAR` the same way as `FIELD_TYPE_CHAR`.
- `CREATE INDEX` and `DROP INDEX` are now implemented through `ALTER TABLE`. `CREATE TABLE` is still the recommended (fast) way to create indexes.
- Added `−−set−variable` option `wait_timeout` to `mysqld`.
- Added time column to `mysqladmin processlist` to show how long a query has taken or how long a thread has slept.
- Added lots of new variables to `show variables` and some new to `show status`.
- Added new type `YEAR`. `YEAR` is stored in 1 byte with allowable values of 0, and 1901 to 2155.
- Added new `DATE` type that is stored in 3 bytes rather than 4 bytes. All new tables are created with the new date type if you don't use the `−−old−protocol` option to `mysqld`.
- Fixed bug in record caches; for some queries, you could get `Error from table handler: #` on some operating systems.
- Added `−−enable−assembler` option to `configure`, for x86 machines (tested on Linux + `gcc`). This will enable assembler functions

for the most important string functions for more speed!

D.3 Modifications de la version 3.21.x

D.3.1 Modifications de la version 3.21.33

- Fixed problem when sending `SIGHUP` to `mysqld`; `mysqld` core dumped when starting from boot on some systems.
- Fixed problem with losing a little memory for some connections.
- `DELETE FROM nom_table` without a `WHERE` condition is now done the long way when you use `LOCK TABLES` or if the table is in use, to avoid race conditions.
- `INSERT INTO TABLE (timestamp_column) VALUES (NULL);` didn't set timestamp.

D.3.2 Modifications de la version 3.21.32

- Fixed some possible race conditions when doing many reopen/close on the same tables under heavy load! This can happen if you execute `mysqladmin refresh` often. This could in some very rare cases corrupt the header of the index file and cause error 126 or 138.
- Fixed fatal bug in `refresh()` when running with the `--skip-locking` option. There was a ``very small'' time gap after a `mysqladmin refresh` when a table could be corrupted if one thread updated a table while another thread did `mysqladmin refresh` and another thread started a new update ont the same table before the first thread had finished. A refresh (or `--flush-tables`) will now not return until all used tables are closed!
- `SELECT DISTINCT` with a `WHERE` clause that didn't match any rows returned a row in some contexts (bug only in 3.21.31).
- `GROUP BY` + `ORDER BY` returned one empty row when no rows where found.
- Fixed a bug in the range optimizer that wrote `Use_count: Wrong count for ...` in the error log file.

D.3.3 Modifications de la version 3.21.31

- Fixed a sign extension problem for the `TINYINT` type on Irix.
- Fixed problem with `LEFT("constant_string",function)`.
- Fixed problem with `FIND_IN_SET()`.
- `LEFT JOIN` core dumped if the second table is used with a constant `WHERE/ON` expression that uniquely identifies one record.
- Fixed problems with `DATE_FORMAT()` and incorrect dates. `DATE_FORMAT()` now ignores `'%'` to make it possible to extend it more easily in the future.

D.3.4 Modifications de la version 3.21.30

- `mysql` now returns an exit code > 0 if the query returned an error.
- Saving of command line history to file in `mysql` client. By Tommy Larsen tommy@mix.hive.no.
- Fixed problem with empty lines that were ignored in `` `mysql.cc' ``.
- Save the pid of the signal handler thread in the pid file instead of the pid of the main thread.
- Added patch by tommy@valley.ne.jp to support Japanese characters SJIS and UJIS.
- Changed `safe_mysqld` to redirect startup messages to `'hostname'.err` instead of `'hostname'.log` to reclaim file space on `mysqladmin refresh`.
- `ENUM` always had the first entry as default value.
- `ALTER TABLE` wrote two entries to the update log.
- `sql_acc()` now closes the `mysql` grant tables after a reload to save table space and memory.
- Changed `LOAD DATA` to use less memory with tables and `BLOB` columns.
- Sorting on a function which made a division / 0 produced a wrong set in some cases.
- Fixed `SELECT` problem with `LEFT()` when using the czech character set.
- Fixed problem in `isamchk`; it couldn't repair a packed table in a very unusual case.
- `SELECT` statements with & or `|` (bit functions) failed on columns with `NULL` values.
- When comparing a field = field, where one of the fields was a part key, only the length of the part key was compared.

D.3.5 Modifications de la version 3.21.29

- `LOCK TABLES` + `DELETE from nom_table` never removed locks properly.
- Fixed problem when grouping on an `OR` function.
- Fixed permission problem with `umask()` and creating new databases.
- Fixed permission problem on result file with `SELECT ... INTO OUTFILE ...`
- Fixed problem in range optimizer (core dump) for a very complex query.
- Fixed problem when using `MIN(integer)` or `MAX(integer)` in `GROUP BY`.
- Fixed bug on Alpha when using integer keys. (Other keys worked on Alpha).

- Fixed bug in `WEEK("XXXX−xx−01")`.

D.3.6 Modifications de la version 3.21.28

- Fixed socket permission (clients couldn't connect to Unix socket on Linux).
- Fixed bug in record caches; for some queries, you could get `Error from table handler: #` on some operating systems.

D.3.7 Modifications de la version 3.21.27

- Added user level lock functions `GET_LOCK(string,timeout)`, `RELEASE_LOCK(string)`.
- Added `opened_tables` to `show status`.
- Changed connect timeout to 3 seconds to make it somewhat harder for crackers to kill `mysqld` through telnet + TCP/IP.
- Fixed bug in range optimizer when using `WHERE key_part_1 >= something AND key_part_2 <= something_else`.
- Changed `configure` for detection of FreeBSD 3.0 9803xx and above
- `WHERE` with string_column_key = constant_string didn't always find all rows if the column had many values differing only with characters of the same sort value (like e and 'e).
- Strings keys looked up with 'ref' were not compared in case−sensitive fashion.
- Added `umask()` to make log files non−readable for normal users.
- Ignore users with old (8−byte) password on startup if not using `−−old−protocol` option to `mysqld`.
- `SELECT` which matched all key fields returned the values in the case of the matched values, not of the found values. (Minor problem.)

D.3.8 Modifications de la version 3.21.26

- `FROM_DAYS(0)` now returns "0000−00−00".
- In `DATE_FORMAT()`, PM and AM were swapped for hours 00 and 12.
- Extended the default maximum key size to 256.
- Fixed bug when using `BLOB`/`TEXT` in `GROUP BY` with many tables.
- An `ENUM` field that is not declared `NOT NULL` has `NULL` as the default value. (Previously, the default value was the first enumeration value.)
- Fixed bug in the join optimizer code when using many part keys on the same key: `INDEX (Organization,Surname(35),Initials(35))`.
- Added some tests to the table order optimizer to get some cases with `SELECT ... FROM many_tables` much faster.
- Added a retry loop around `accept()` to possibly fix some problems on some Linux machines.

D.3.9 Modifications de la version 3.21.25

- Changed `typedef 'string'` to `typedef 'my_string'` for better portability.
- You can now kill threads that are waiting on a disk full condition.
- Fixed some problems with UDF functions.
- Added long options to `isamchk`. Try `isamchk −−help`.
- Fixed a bug when using 8 bytes long (alpha); `filesort()` didn't work. Affects `DISTINCT`, `ORDER BY` and `GROUP BY` on 64−bit processors.

D.3.10 Modifications de la version 3.21.24

- Dynamic loadable functions. Based on source from Alexis Mikhailov.
- You couldn't delete from a table if no one had done a `SELECT` on the table.
- Fixed problem with range optimizer with many `OR` operators on key parts inside each other.
- Recoded `MIN()` and `MAX()` to work properly with strings and `HAVING`.
- Changed default umask value for new files from `0664` to `0660`.
- Fixed problem with `LEFT JOIN` and constant expressions in the `ON` part.
- Added Italian error messages from brenno@3cord.philips.nl.
- `configure` now works better on OSF1 (tested on 4.0D).
- Added hooks to allow `LIKE` optimization with international character support.
- Upgraded `DBI` to 0.93.

D.3.11 Modifications de la version 3.21.23

- The following symbols are now reserved words: `TIME`, `DATE`, `TIMESTAMP`, `TEXT`, `BIT`, `ENUM`, `NO`, `ACTION`, `CHECK`, `YEAR`, `MONTH`, `DAY`, `HOUR`, `MINUTE`, `SECOND`, `STATUS`, `VARIABLES`.
- Setting a `TIMESTAMP` to `NULL` in `LOAD DATA INFILE ...` didn't set the current time for the `TIMESTAMP`.

- Fix `BETWEEN` to recognize binary strings. Now `BETWEEN` is case sensitive.
- Added `--skip-thread-priority` option to `mysqld`, for systems where `mysqld`'s thread scheduling doesn't work properly (BSDI 3.1).
- Added ODBC functions `DAYNAME()` and `MONTHNAME()`.
- Added function `TIME_FORMAT()`. This works like `DATE_FORMAT()`, but takes a time string (`'HH:MM:DD'`) as argument.
- Fixed unlikely(?) key optimizer bug when using `OR`s of key parts inside `AND`s.
- Added command `variables` to `mysqladmin`.
- A lot of small changes to the binary releases.
- Fixed a bug in the new protocol from ***MySQL*** 3.21.20.
- Changed `ALTER TABLE` to work with Win32 (Win32 can't rename open files). Also fixed a couple of small bugs in the Win32 version.
- All standard ***MySQL*** clients are now ported to MySQL−Win32.
- ***MySQL*** can now be started as a service on NT.

D.3.12 Modifications de la version 3.21.22

- Starting with this version, all ***MySQL*** distributions will be configured, compiled and tested with `crash-me` and the benchmarks on the following platforms: SunOS 5.6 sun4u, SunOS 5.5.1 sun4u, SunOS 4.14 sun4c, SunOS 5.6 i86pc, Irix 6.3 mips5k, HP−UX 10.20 hppa, AIX 4.2.1 ppc, OSF1 V4.0 alpha, FreeBSD 2.2.2 i86pc and BSDI 3.1 i386.
- Fix `COUNT(*)` problems when the `WHERE` clause didn't match any records. (Bug from 3.21.17.)
- Removed that `NULL = NULL` is true. Now you must use `IS NULL` or `IS NOT NULL` to test whether or not a value is `NULL`. (This is according to ANSI SQL but may break old applications that are ported from `mSQL`.) You can get the old behavior by compiling with `-DmSQL_COMPLIANT`.
- Fixed bug that core dumped when using many `LEFT OUTER JOIN` clauses.
- Fixed bug in `ORDER BY` on string formula with possible `NULL` values.
- Fixed problem in range optimizer when using <= on sub index.
- Added functions `DAYOFYEAR()`, `DAYOFMONTH()`, `MONTH()`, `YEAR()`, `WEEK()`, `QUARTER()`, `HOUR()`, `MINUTE()`, `SECOND()` and `FIND_IN_SET()`.
- Added command `SHOW VARIABLES`.
- Added support of ``long constant strings'' from ANSI SQL:

```
mysql62; SELECT 'first ' 'second';       -62; 'first second'
```

- Upgraded mSQL−Mysql−modules to 1.1825.
- Upgraded `mysqlaccess` to 2.02.
- Fixed problem with Russian character set and `LIKE`.
- Ported to OpenBSD 2.1.
- New Dutch error messages.

D.3.13 Modifications de la version 3.21.21a

- Configure changes for some operating systems.

D.3.14 Modifications de la version 3.21.21

- Fixed optimizer bug when using `WHERE data_field = date_field2 AND date_field2 = constant`.
- Added command `SHOW STATUS`.
- Removed `` `manual.ps' `` from the source distribution to make it smaller.

D.3.15 Modifications de la version 3.21.20

- Changed the maximum table name and column name lengths from 32 to 64.
- Aliases can now be of ``any'' length.
- Fixed `mysqladmin stat` to return the right number of queries.
- Changed protocol (downward compatible) to mark if a column has the `AUTO_INCREMENT` attribute or is a `TIMESTAMP`. This is needed for the new Java driver.
- Added Hebrew sorting order by Zeev Suraski.
- Solaris 2.6: Fixed `configure` bugs and increased maximum table size from 2G to 4G.

D.3.16 Modifications de la version 3.21.19

- Upgraded `DBD` to 1823. This version implements `mysql_use_result` in `DBD-Mysql`.
- Benchmarks updated for empress (by Luuk).
- Fixed a case of slow range searching.

- Configure fixes (`Docs' directory).
- Added function REVERSE() (by Zeev Suraski).

D.3.17 Modifications de la version 3.21.18

- Issue error message if client C functions are called in wrong order.
- Added automatic reconnect to the `libmysql.c' library. If a write command fails, an automatic reconnect is done.
- Small sort sets no longer use temporary files.
- Upgraded DBI to 0.91.
- Fixed a couple of problems with LEFT OUTER JOIN.
- Added CROSS JOIN syntax. CROSS is now a reserved word.
- Recoded yacc/bison stack allocation to be even safer and to allow ***MySQL*** to handle even bigger expressions.
- Fixed a couple of problems with the update log.
- ORDER BY was slow when used with key ranges.

D.3.18 Modifications de la version 3.21.17

- Changed documentation string of --with-unix-socket-path to avoid confusion.
- Added ODBC and ANSI SQL style LEFT OUTER JOIN.
- The following are new reserved words: LEFT, NATURAL, USING.
- The client library now uses the value of the environment variable MYSQL_HOST as the default host if it's defined.
- SELECT column, SUM(expr) now returns NULL for column when there are matching rows.
- Fixed problem with comparing binary strings and BLOBs with ASCII characters over 127.
- Fixed lock problem: when freeing a read lock on a table with multiple read locks, a thread waiting for a write lock would have been given the lock. This shouldn't affect data integrity, but could possibly make mysqld restart if one thread was reading data that another thread modified.
- LIMIT offset,count didn't work in INSERT ... SELECT.
- Optimized key block caching. This will be quicker than the old algorithm when using bigger key caches.

D.3.19 Modifications de la version 3.21.16

- Added ODBC 2.0 & 3.0 functions POWER(), SPACE(), COT(), DEGREES(), RADIANS(), ROUND(2 arg) and TRUNCATE().
- ***WARNING: INCOMPATIBLE CHANGE!!*** LOCATE() parameters were swapped according to ODBC standard. Fixed.
- Added function TIME_TO_SEC().
- In some cases, default values were not used for NOT NULL fields.
- Timestamp wasn't always updated properly in UPDATE SET ... statements.
- Allow empty strings as default values for BLOB and TEXT, to be compatible with mysqldump.

D.3.20 Modifications de la version 3.21.15

- ***WARNING: INCOMPATIBLE CHANGE!!*** mysqlperl is now from Msql−Mysql−modules. This means that connect() now takes host, database, user, password arguments! The old version took host, database, password, user.
- Allow DATE '1997−01−01', TIME '12:10:10' and TIMESTAMP '1997−01−01 12:10:10' formats required by ANSI SQL. ***WARNING: INCOMPATIBLE CHANGE!!*** This has the unfortunate side−effect that you no longer can have columns named DATE, TIME or TIMESTAMP. :(Old columns can still be accessed through tablename.columnname!)
- Changed Makefiles to hopefully work better with BSD systems. Also, `manual.dvi' is now included in the distribution to avoid having stupid make programs trying to rebuild it.
- readline library upgraded to version 2.1.
- A new sortorder german−1. That is a normal ISO−Latin1 with a german sort order.
- Perl DBI/DBD is now included in the distribution. DBI is now the recommended way to connect to ***MySQL*** from Perl.
- New portable benchmark suite with DBD, with test results from mSQL 2.0.3, ***MySQL***, PostgreSQL 6.2.1 and Solid server 2.2.
- crash−me is now included with the benchmarks; This is a Perl program designed to find as many limits as possible in a SQL server. Tested with mSQL, PostgreSQL, Solid and ***MySQL***.
- Fixed bug in range−optimizer that crashed ***MySQL*** on some queries.
- Table and column name completion for mysql command line tool, by Zeev Suraski and Andi Gutmans.
- Added new command REPLACE that works like INSERT but replaces conflicting records with the new record. REPLACE INTO TABLE ... SELECT ... works also.
- Added new commands CREATE DATABASE nom_base_de_donnees and DROP DATABASE nom_base_de_donnees.
- Added RENAME option to ALTER TABLE: ALTER TABLE name RENAME AS new_name.
- make_binary_distribution now includes `libgcc.a' in `libmysqlclient.a'. This should make linking work for people who don't have gcc.
- Changed net_write() to my_net_write() because of a name conflict with Sybase.
- New function DAYOFWEEK() compatible with ODBC.
- Stack checking and bison memory overrun checking to make ***MySQL*** safer with weird queries.

D.3.21 Modifications de la version 3.21.14b

- Fixed a couple of small `configure` problems on some platforms.

D.3.22 Modifications de la version 3.21.14a

- Ported to SCO Openserver 5.0.4 with FSU Pthreads.
- HP−UX 10.20 should work.
- Added new function `DATE_FORMAT()`.
- Added `NOT IN`.
- Added automatic removal of 'ODBC function conversions': `{fn now() }`
- Handle ODBC 2.50.3 option flags.
- Fixed comparison of `DATE` and `TIME` values with `NULL`.
- Changed language name from germany to german to be consistent with the other language names.
- Fixed sorting problem on functions returning a `FLOAT`. Previously, the values were converted to `INT`s before sorting.
- Fixed slow sorting when sorting on key field when using `key_column=constant`.
- Sorting on calculated `DOUBLE` values sorted on integer results instead.
- `mysql` no longer needs a database argument.
- Changed the place where `HAVING` should be. According to ANSI, it should be after `GROUP BY` but before `ORDER BY`. ***MySQL*** 3.20 incorrectly had it last.
- Added Sybase command `USE DATABASE` to start using another database.
- Added automatic adjusting of number of connections and table cache size if the maximum number of files that can be opened is less than needed. This should fix that `mysqld` doesn't crash even if you haven't done a `ulimit -n 256` before starting `mysqld`.
- Added lots of limit checks to make it safer when running with too little memory or when doing weird queries.

D.3.23 Modifications de la version 3.21.13

- Added retry of interrupted reads and clearing of `errno`. This makes Linux systems much safer!
- Fixed locking bug when using many aliases on the same table in the same `SELECT`.
- Fixed bug with `LIKE` on number key.
- New error message so you can check whether the connection was lost while the command was running or whether the connection was down from the start.
- Added `--table` option to `mysql` to print in table format. Moved time and row information after query result. Added automatic reconnect of lost connections.
- Added `!=` as a synonym for `<>`.
- Added function `VERSION()` to make easier logs.
- New multi−user test `` `tests/fork_test.pl' `` to put some strain on the thread library.

D.3.24 Modifications de la version 3.21.12

- Fixed `ftruncate()` call in MIT−pthreads. This made `isamchk` destroy the `` `.ISM' `` files on (Free)BSD 2.x systems.
- Fixed broken __P_ patch in MIT−pthreads.
- Many memory overrun checks. All string functions now return `NULL` if the returned string should be longer than `max_allowed_packet` bytes.
- Changed the name of the `INTERVAL` type to `ENUM`, because `INTERVAL` is used in ANSI SQL.
- In some cases, doing a `JOIN` + `GROUP` + `INTO OUTFILE`, the result wasn't grouped.
- `LIKE` with `'_'` as last character didn't work. Fixed.
- Added extended ANSI SQL `TRIM()` function.
- Added `CURTIME()`.
- Added `ENCRYPT()` function by Zeev Suraski.
- Fixed better `FOREIGN KEY` syntax skipping. New reserved words: `MATCH`, `FULL`, `PARTIAL`.
- `mysqld` now allows IP number and hostname to the `--bind-address` option.
- Added `SET OPTION CHARACTER SET cp1251_koi8` to enable conversions of data to/from cp1251_koi8.
- Lots of changes for Win95 port. In theory, this version should now be easily portable to Win95.
- Changed the `CREATE COLUMN` syntax of `NOT NULL` columns to be after the `DEFAULT` value, as specified in the ANSI SQL standard. This will make `mysqldump` with `NOT NULL` and default values incompatible with ***MySQL*** 3.20.
- Added many function name aliases so the functions can be used with ODBC or ANSI SQL92 syntax.
- Fixed syntax of `ALTER TABLE nom_table ALTER COLUMN nom_colonne SET DEFAULT NULL`.
- Added `CHAR` and `BIT` as synonyms for `CHAR(1)`.
- Fixed core dump when updating as a user who has only ***select*** privilege.
- `INSERT ... SELECT ... GROUP BY` didn't work in some cases. An `Invalid use of group function` error occurred.
- When using `LIMIT`, `SELECT` now always uses keys instead of record scan. This will give better performance on `SELECT` and a `WHERE` that matches many rows.
- Added Russian error messages.

D.3.25 Modifications de la version 3.21.11

- Configure changes.
- ***MySQL*** now works with the new thread library on BSD/OS 3.0.
- Added new group functions `BIT_OR()` and `BIT_AND()`.
- Added compatibility functions `CHECK` and `REFERENCES`. `CHECK` is now a reserved word.
- Added `ALL` option to `GRANT` for better compatibility. (`GRANT` is still a dummy function.)
- Added partly−translated dutch messages.
- Fixed bug in `ORDER BY` and `GROUP BY` with `NULL` columns.
- Added function `last_insert_id()` to retrieve last `AUTO_INCREMENT` value. This is intended for clients to ODBC that can't use the `mysql_insert_id()` API function, but can be used by any client.
- Added `--flush-logs` option to `mysqladmin`.
- Added command `STATUS` to `mysql`.
- Fixed problem with `ORDER BY`/`GROUP BY` because of bug in `gcc`.
- Fixed problem with `INSERT ... SELECT ... GROUP BY`.

D.3.26 Modifications de la version 3.21.10

- New `mysqlaccess`.
- `CREATE` now supports all ODBC types and the `mSQL TEXT` type. All ODBC 2.5 functions are also supported (added `REPEAT`). This provides better portability.
- Added text types `TINYTEXT`, `TEXT`, `MEDIUMTEXT` and `LONGTEXT`. These are actually `BLOB`types, but all searching is done in case−insensitive fashion.
- All old `BLOB` fields are now `TEXT` fields. This only changes that all searching on strings is done in case−sensitive fashion. You must do an `ALTER TABLE` and change the field type to `BLOB` if you want to have tests done in case−sensitive fashion.
- Fixed some `configure` issues.
- Made the locking code a bit safer. Fixed very unlikely deadlock situation.
- Fixed a couple of bugs in the range optimizer. Now the new range benchmark `test-select` works.

D.3.27 Modifications de la version 3.21.9

- Added `--enable-unix-socket=pathname` option to `configure`.
- Fixed a couple of portability problems with include files.
- Fixed bug in range calculation that could return empty set when searching on multiple key with only one entry (very rare).
- Most things ported to FSU Pthreads, which should allow ***MySQL*** to run on SCO. 4.11.12 SCO.

D.3.28 Modifications de la version 3.21.8

- Works now in Solaris 2.6.
- Added handling of calculation of `SUM()` functions. For example, you can now use `SUM(column)/COUNT(column)`.
- Added handling of trigometric functions: `PI()`, `ACOS()`, `ASIN()`, `ATAN()`, `COS()`, `SIN()` and `TAN()`.
- New languages: norwegian, norwegian−ny and portuguese.
- Fixed parameter bug in `net_print()` in `` `procedure.cc' ``.
- Fixed a couple of memory leaks.
- Now allow also the old `SELECT ... INTO OUTFILE` syntax.
- Fixed bug with `GROUP BY` and `SELECT` on key with many values.
- `mysql_fetch_lengths()` sometimes returned incorrect lengths when you used `mysql_use_result()`. This affected at least some cases of `mysqldump --quick`.
- Fixed bug in optimization of `WHERE const op field`.
- Fixed problem when sorting on `NULL` fields.
- Fixed a couple of 64−bit (Alpha) problems.
- Added `--pid-file=#` option to `mysqld`.
- Added date formatting to `FROM_UNIXTIME()`, originally by Zeev Suraski.
- Fixed bug in `BETWEEN` in range optimizer (Did only test = of the first argument).
- Added machine−dependent files for MIT−pthreads i386−SCO. There is probably more to do to get this to work on SCO 3.5.

D.3.29 Modifications de la version 3.21.7

- Changed `` `Makefile.am' `` to take advantage of Automake 1.2.
- Added the beginnings of a benchmark suite.
- Added more secure password handling.
- Added new client function `mysql_errno()`, to get the error number of the error message. This makes error checking in the client much easier. This makes the new server incompatible with the 3.20.x server when running without `--old-protocol`. The client code is

backward compatible. More information can be found in the `README' file!
- Fixed some problems when using very long, illegal names.

D.3.30 Modifications de la version 3.21.6

- Fixed more portability issues (incorrect `sigwait` and `sigset` defines).
- `configure` should now be able to detect the last argument to `accept()`.

D.3.31 Modifications de la version 3.21.5

- Should now work with FreeBSD 3.0 if used with `FreeBSD−3.0−libc_r−1.0.diff', which can be found at http://www.mysql.com/Download/Patches.
- Added new option `−O tmp_table_size=#` to `mysqld`.
- New function `FROM_UNIXTIME(timestamp)` which returns a date string in 'YYYY−MM−DD HH:MM:DD' format.
- New function `SEC_TO_TIME(seconds)` which returns a string in 'HH:MM:SS' format.
- New function `SUBSTRING_INDEX()`, originally by Zeev Suraski.

D.3.32 Modifications de la version 3.21.4

- Should now configure and compile on OSF1 4.0 with the DEC compiler.
- Configuration and compilation on BSD/OS 3.0 works, but due to some bugs in BSD/OS 3.0, `mysqld` doesn't work on it yet.
- Configuration and compilation on FreeBSD 3.0 works, but I couldn't get `pthread_create` to work.

D.3.33 Modifications de la version 3.21.3

- Added reverse check lookup of hostnames to get better security.
- Fixed some possible buffer overflows if filenames that are too long are used.
- `mysqld` doesn't accept hostnames that start with digits followed by a `'.'`, because the hostname may look like an IP number.
- Added `−−skip−networking` option to `mysqld`, to only allow socket connections. (This will not work with MIT−pthreads!)
- Added check of too long table names for alias.
- Added check if database name is okay.
- Added check if too long table names.
- Removed incorrect `free()` that killed the server on `CREATE DATABASE` or `DROP DATABASE`.
- Changed some `mysqld −O` options to better names.
- Added `−O join_cache_size=#` option to `mysqld`.
- Added `−O max_join_size=#` option to `mysqld`, to be able to set a limit how big requêtes(in this case big = slow) one should be able to handle without specifying `SET OPTION SQL_BIG_SELECTS=1`. A # = is about 10 examined records. The default is ``unlimited".
- When comparing a `TIME`, `DATE`, `DATETIME` or `TIMESTAMP` column to a constant, the constant is converted to a time value before performing the comparison. This will make it easier to get ODBC (particularly Access97) to work with the above types. It should also make dates easier to use and the comparisons should be quicker than before.
- Applied patch from Jochen Wiedmann that allows `query()` in `mysqlperl` to take a query with `\0` in it.
- Storing a timestamp with a 2−digit year (`YYMMDD`) didn't work.
- Fix that timestamp wasn't automatically updated if set in an `UPDATE` clause.
- Now the automatic timestamp field is the FIRST timestamp field.
- `SELECT * INTO OUTFILE`, which didn't correctly if the outfile already existed.
- `mysql` now shows the thread ID when starting or doing a reconnect.
- Changed the default sort buffer size from 2M to 1M.

D.3.34 Modifications de la version 3.21.2

- The range optimizer is coded, but only 85% tested. It can be enabled with `−−new`, but it crashes core a lot yet...
- More portable. Should compile on AIX and alpha−digital. At least the `isam` library should be relatively 64−bit clean.
- New `isamchk` which can detect and fix more problems.
- New options for `isamlog`.
- Using new version of Automake.
- Many small portability changes (from the AIX and alpha−digital port) Better checking of pthread(s) library.
- czech error messages by snajdr@pvt.net.
- Decreased size of some buffers to get fewer problems on systems with little memory. Also added more checks to handle ``out of memory" problems.
- `mysqladmin`: you can now do `mysqladmin kill 5,6,7,8` to kill multiple threads.
- When the maximum connection limit is reached, one extra connection by a user with the ***PROCESS_ACL*** privilege is granted.
- Added `−O backlog=#` option to `mysqld`.

- Increased maximum packet size from 512K to 1024K for client.
- Almost all of the function code is now tested in the internal test suite.
- `ALTER TABLE` now returns warnings from field conversions.
- Port changed to 3306 (got it reserved from ISI).
- Added a fix for Visual FoxBase so that any schema name from a table specification is automatically removed.
- New function `ASCII()`.
- Removed function `BETWEEN(a,b,c)`. Use the standard ANSI synax instead: `expr BETWEEN expr AND expr`.
- ***MySQL*** no longer has to use an extra temporary table when sorting on functions or `SUM()` functions.
- Fixed bug that you couldn't use `nom_table.field_name` in `UPDATE`.
- Fixed `SELECT DISTINCT` when using 'hidden group'. For example:

```
mysql62; SELECT DISTINCT MOD(some_field,10) FROM test
           GROUP BY some_field;
```

Note: `some_field` is normally in the `SELECT` part. ANSI SQL should require it.

D.3.35 Modifications de la version 3.21.0

- New keywords used: `INTERVAL`, `EXPLAIN`, `READ`, `WRITE`, `BINARY`.
- Added ODBC function `CHAR(num,...)`.
- New operator `IN`. This uses a binary search to find a match.
- New command `LOCK TABLES nom_table [AS alias] {READ|WRITE} ...`
- Added `--log-update` option to `mysqld`, to get a log suitable for incremental updates.
- New command `EXPLAIN SELECT ...` to get information about how the optimizer will do the join.
- For easier client code, the client should no longer use `FIELD_TYPE_TINY_BLOB`, `FIELD_TYPE_MEDIUM_BLOB`, `FIELD_TYPE_LONG_BLOB` or `FIELD_TYPE_VAR_STRING` (as previously returned by `mysql_list_fields`). You should instead only use `FIELD_TYPE_BLOB` or `FIELD_TYPE_STRING`. If you want exact types, you should use the command `SHOW FIELDS`.
- Added varbinary syntax: `0x######` which can be used as a string (default) or a number.
- `FIELD_TYPE_CHAR` is renamed to `FIELD_TYPE_TINY`.
- Changed all fields to C++ classes.
- Removed FORM struct.
- Fields with `DEFAULT` values no longer need to be `NOT NULL`.
- New field types:

  ENUM
  : A string which can take only a couple of defined values. The value is stored as a 1−3 byte number that is mapped automatically to a string. This is sorted according to string positions!

  SET
  : A string which may have one or many string values separated with ','. The string is stored as a 1−, 2−, 3−, 4− or 8−byte number where each bit stands for a specific set member. This is sorted according to the unsigned value of the stored packed number.
- Now all function calculation is done with `double` or `long long`. This will provide the full 64−bit range with bit functions and fix some conversions that previously could result in precision losses. One should avoid using `unsigned long long` columns with full 64−bit range (numbers bigger than 9223372036854775807) because calculations are done with `signed long long`.
- `ORDER BY` will now put `NULL` field values first. `GROUP BY` will also work with `NULL` values.
- Full `WHERE` with expressions.
- New range optimizer that can resolve ranges when some keypart prefix is constant. Example:

```
mysql62; SELECT * FROM nom_table
           WHERE key_part_1="customer"
           AND key_part_262;=10 AND key_part_260;=10;
```

21.3 Modifications de la version 3.20.x

Changes from 3.20.18 to 3.20.32b are not documented here since the 3.21 release branched here. And the relevant changes are also documented as changes to the 3.21 version.

D.3.36 Modifications de la version 3.20.18

- Added `-p#` (remove # directories from path) to `isamlog`. All files are written with a relative path from the database directory Now `mysqld` shouldn't crash on shutdown when using the `--log-isam` option.
- New `mysqlperl` version. It is now compatible with `msqlperl-0.63`.
- New `DBD` module available at http://www.mysql.com/Contrib site.
- Added group function `STD()` (standard deviation).
- The `mysqld` server is now compiled by default without debugging information. This will make the daemon smaller and faster.
- Now one usually only has to specify the `--basedir` option to `mysqld`. All other paths are relative in a normal installation.
- `BLOB` columns sometimes contained garbage when used with a `SELECT` on more than one table and `ORDER BY`.

- Fixed that calculations that are not in `GROUP BY` work as expected (ANSI SQL extension). Example:
  `mysql62; SELECT id,id+1 FROM table GROUP BY id;`

- The test of using `MYSQL_PWD` was reversed. Now `MYSQL_PWD` is enabled as default in the default release.
- Fixed conversion bug which caused `mysqld` to core dump with Arithmetic error on Sparc−386.
- Added `−−unbuffered` option to `mysql`, for new `mysqlaccess`.
- When using overlapping (unnecessary) keys and join over many tables, the optimizer could get confused and return 0 records.

D.3.37 Modifications de la version 3.20.17

- You can now use `BLOB` columns and the functions `IS NULL` and `IS NOT NULL` in the `WHERE` clause.
- All communication packets and row buffers are now allocated dynamically on demand. The default value of `max_allowed_packet` is now 64K for the server and 512K for the client. This is mainly used to catch incorrect packets that could trash all memory. The server limit may be changed when it is started.
- Changed stack usage to use less memory.
- Changed `safe_mysqld` to check for running daemon.
- The `ELT()` function is renamed to `FIELD()`. The new `ELT()` function returns a value based on an index: `FIELD()` is the inverse of `ELT()` Example: `ELT(2,"A","B","C")` returns `"B"`. `FIELD("B","A","B","C")` returns 2.
- `COUNT(field)`, where `field` could have a `NULL` value, now works.
- A couple of bugs fixed in `SELECT ... GROUP BY`.
- Fixed memory overrun bug in `WHERE` with many unoptimizable brace levels.
- Fixed some small bugs in the grant code.
- If hostname isn't found by `get_hostname`, only the IP is checked. Previously, you got `Access denied`.
- Inserts of timestamps with values didn't always work.
- `INSERT INTO ... SELECT ... WHERE` could give the error `Duplicated field`.
- Added some tests to `safe_mysqld` to make it ``safer''.
- `LIKE` was case sensitive in some places and case insensitive in others. Now `LIKE` is always case insensitive.
- `` `mysql.cc' ``: Allow `'#'` anywhere on the line.
- New command `SET OPTION SQL_SELECT_LIMIT=#`. See the FAQ for more details.
- New version of the `mysqlaccess` script.
- Change `FROM_DAYS()` and `WEEKDAY()` to also take a full `TIMESTAMP` or `DATETIME` as argument. Before they only took a number of type `YYYYMMDD` or `YYMMDD`.
- Added new function `UNIX_TIMESTAMP(timestamp_column)`.

D.3.38 Modifications de la version 3.20.16

- More changes in MIT−pthreads to get them safer. Fixed also some link bugs at least in SunOS.
- Changed `mysqld` to work around a bug in MIT−pthreads. This makes multiple small `SELECT` operations 20 times faster. Now `lock_test.pl` should work.
- Added `mysql_FetchHash(handle)` to `mysqlperl`.
- The `mysqlbug` script is now distributed built to allow for reporting bugs that appear during the build with it.
- Changed `` `libmysql.c' `` to prefer `getpwuid()` instead of `cuserid()`.
- Fixed bug in `SELECT` optimizer when using many tables with the same column used as key to different tables.
- Added new latin2 and Russian KOI8 character tables.
- Added support for a dummy `GRANT` command to satisfy Powerbuilder.

D.3.39 Modifications de la version 3.20.15

- Fixed fatal bug `packets out of order` when using MIT−pthreads.
- Removed possible loop when a thread waits for command from client and `fcntl()` fails. Thanks to Mike Bretz for finding this bug.
- Changed alarm loop in `` `mysqld.cc' `` because shutdown didn't always succeed in Linux.
- Removed use of `termbits` from `` `mysql.cc' ``. This conflicted with `glibc` 2.0.
- Fixed some syntax errors for at least BSD and Linux.
- Fixed bug when doing a `SELECT` as superuser without a database.
- Fixed bug when doing `SELECT` with group calculation to outfile.

D.3.40 Modifications de la version 3.20.14

- If one gives `−p` or `−−password` option to `mysql` without an argument, the user is solicited for the password from the tty.
- Added default password from `MYSQL_PWD` (by Elmar Haneke).
- Added command `kill` to `mysqladmin` to kill a specific ***MySQL*** thread.
- Sometimes when doing a reconnect on a down connection this succeeded first on second try.
- Fixed adding an `AUTO_INCREMENT` key with `ALTER TABLE`.

- AVG() gave too small value on some SELECTs with GROUP BY and ORDER BY.
- Added new DATETIME type (by Giovanni Maruzzelli
- Fixed that define DONT_USE_DEFAULT_FIELDS works.
- Changed to use a thread to handle alarms instead of signals on Solaris to avoid race conditions.
- Fixed default length of signed numbers. (George Harvey
- Allow anything for CREATE INDEX.
- Add prezeros when packing numbers to DATE, TIME and TIMESTAMP.
- Fixed a bug in OR of multiple tables (gave empty set).
- Added many patches to MIT−pthreads. This fixes at least one lookup bug.

D.3.41 Modifications de la version 3.20.13

- Added ANSI SQL94 DATE and TIME types.
- Fixed bug in SELECT with AND−OR levels.
- Added support for Slovenian characters. The `Contrib' directory contains source and instructions for adding other character sets.
- Fixed bug with LIMIT and ORDER BY.
- Allow ORDER BY and GROUP BY on items that aren't in the SELECT list. (Thanks to Wim Bonis bonis@kiss.de, for pointing this out.)
- Allow setting of timestamp values in INSERT.
- Fixed bug with SELECT ... WHERE ... = NULL.
- Added changes for glibc 2.0. To get glibc to work, you should add the `gibc−2.0−sigwait−patch' before compiling glibc.
- Fixed bug in ALTER TABLE when changing a NOT NULL field to allow NULL values.
- Added some ANSI92 synonyms as field types to CREATE TABLE. CREATE TABLE now allows FLOAT(4) and FLOAT(8) to mean FLOAT and DOUBLE.
- New utility program mysqlaccess by Yves.Carlier@rug.ac.be. This program shows the access rights for a specific user and the grant rows that determine this grant.
- Added WHERE const op field (by bonis@kiss.de).

D.3.42 Modifications de la version 3.20.11

- When using SELECT ... INTO OUTFILE, all temporary tables are ISAM instead of HEAP to allow big dumps.
- Changed date functions to be string functions. This fixed some ``funny'' side effects when sorting on dates.
- Extended ALTER TABLE according to SQL92.
- Some minor compability changes.
- Added −−port and −−socket options to all utility programs and mysqld.
- Fixed MIT−pthreads readdir_r(). Now mysqladmin create database and mysqladmin drop database should work.
- Changed MIT−pthreads to use our tempnam(). This should fix the ``sort aborted'' bug.
- Added sync of records count in sql_update. This fixed slow updates on first connection. (Thanks to Vaclav Bittner for the test.)

D.3.43 Modifications de la version 3.20.10

- New insert type: INSERT INTO ... SELECT ...
- MEDIUMBLOB fixed.
- Fixed bug in ALTER TABLE and BLOBs.
- SELECT ... INTO OUTFILE now creates the file in the current database directory.
- DROP TABLE now can take a list of tables.
- Oracle synonym DESCRIBE (DESC).
- Changes to make_binary_distribution.
- Added some comments to installation instructions about configure's C++ link test.
- Added −−without−perl option to configure.
- Lots of small portability changes.

D.3.44 Modifications de la version 3.20.9

- ALTER TABLE didn't copy null bit. As a result, fields that were allowed to have NULL values were always NULL.
- CREATE didn't take numbers as DEFAULT.
- Some compatibility changes for SunOS.
- Removed `config.cache' from old distribution.

D.3.45 Modifications de la version 3.20.8

- Fixed bug with ALTER TABLE and multi−part keys.

D.3.46 Modifications de la version 3.20.7

- New commands: `ALTER TABLE`, `SELECT ... INTO OUTFILE` and `LOAD DATA INFILE`.
- New function: `NOW()`.
- Added new field ***file_priv*** to `mysql/user` table.
- New script `add_file_priv` which adds the new field ***file_priv*** to the `user` table. This script must be executed if you want to use the new `SELECT ... INTO` and `LOAD DATA INFILE ...` commands with a version of ***MySQL*** earlier than 3.20.7.
- Fixed bug in locking code, which made `lock_test.pl` test fail.
- New files `` `NEW' `` and `` `BUGS' ``.
- Changed `` `select_test.c' `` and `` `insert_test.c' `` to include `` `config.h' ``.
- Added command `status` to `mysqladmin` for short logging.
- Increased maximum number of keys to 16 and maximum number of key parts to 15.
- Use of sub keys. A key may now be a prefix of a string field.
- Added `−k` option to `mysqlshow`, to get key information for a table.
- Added long options to `mysqldump`.

D.3.47 Modifications de la version 3.20.6

- Portable to more systems because of MIT−pthreads, which will be used automatically if `configure` cannot find a `−lpthreads` library.
- Added GNU−style long options to almost all programs. Test with **`program --help`**.
- Some shared library support for Linux.
- The FAQ is now in `` `.texi' `` format and is available in `` `.html' ``, `` `.txt' `` and `` `.ps' `` formats.
- Added new SQL function `RAND([init])`.
- Changed `sql_lex` to handle `\0` unquoted, but the client can't send the query through the C API, because it takes a str pointer. You must use `mysql_real_query()` to send the query.
- Added API function `mysql_get_client_info()`.
- `mysqld` now uses the `N_MAX_KEY_LENGTH` from `` `nisam.h' `` as the maximum allowed key length.
- The following now works:

```
mysql62; SELECT filter_nr,filter_nr FROM filter ORDER BY filter_nr;
```

  Previously, this resulted in the error: `Column: 'filter_nr' in order clause is ambiguous`.

- `mysql` now outputs `'\0'`, `'\t'`, `'\n'` and `'\\'` when encountering ASCII 0, tab, newline or `'\'` while writing tab−separated output. This is to allow printing of binary data in a portable format. To get the old behavior, use `−r` (or `--raw`).
- Added german error messages (60 of 80 error messages translated).
- Added new API function `mysql_fetch_lengths(MYSQL_RES *)`, which returns an array of of column lengths (of type `uint`).
- Fixed bug with `IS NULL` in `WHERE` clause.
- Changed the optimizer a little to get better results when searching on a key part.
- Added `SELECT` option `STRAIGHT_JOIN` to tell the optimizer that it should join tables in the given order.
- Added support for comments starting with `'--'` in `` `mysql.cc' `` (Postgres syntax).
- You can have `SELECT` expressions and table columns in a `SELECT` which are not used in the group part. This makes it efficient to implement lookups. The column that is used should be a constant for each group because the value is calculated only once for the first row that is found for a group.

```
mysql62; SELECT id,lookup.text,sum(*) FROM test,lookup
         WHERE test.id=lookup.id GROUP BY id;
```

- Fixed bug in `SUM(function)` (could cause a core dump).
- Changed `AUTO_INCREMENT` placement in the SQL query:

```
INSERT into table (auto_field) values (0);
```

  inserted 0, but it should insert an `AUTO_INCREMENT` value.

- `` `mysqlshow.c' ``: Added number of records in table. Had to change the client code a little to fix this.
- `mysql` now allows doubled `''` or `""` within strings for embedded `'` or `"`.
- New math functions: `EXP()`, `LOG()`, `SQRT()`, `ROUND()`, `CEILING()`.

D.3.48 Modifications de la version 3.20.3

- The `configure` source now compiles a thread−free client library `−lmysqlclient`. This is the only library that needs to be linked with client applications. When using the binary releases, you must link with `−lmysql −lmysys −ldbug −lstrings` as before.
- New `readline` library from `bash−2.0`.
- LOTS of small changes to `configure` and makefiles (and related source).
- It should now be possible to compile in another directory using `VPATH`. Tested with GNU Make 3.75.

- `safe_mysqld` and `mysql.server` changed to be more compatible between the source and the binary releases.
- `LIMIT` now takes one or two numeric arguments. If one argument is given, it indicates the maximum number of rows in a result. If two arguments are given, the first argument indicates the offset of the first row to return, the second is the maximum number of rows. With this it's easy to do a poor man's next page/previous page WWW application.
- Changed name of SQL function `FIELDS()` to `ELT()`. Changed SQL function `INTERVALL()` to `INTERVAL()`.
- Made `SHOW COLUMNS` a synonym for `SHOW FIELDS`. Added compatibility syntax `FRIEND KEY` to `CREATE TABLE`. In ***MySQL***, this creates a non−unique key on the given columns.
- Added `CREATE INDEX` and `DROP INDEX` as compatibility functions. In ***MySQL***, `CREATE INDEX` only checks if the index exists and issues an error if it doesn't exist. `DROP INDEX` always succeeds.
- `` `mysqladmin.c' ``: added client version to version information.
- Fixed core dump bug in `sql_acl` (core on new connection).
- Removed `host`, `user` and `db` tables from database `test` in the distribution.
- `FIELD_TYPE_CHAR` can now be signed (−128 − 127) or unsigned (0 − 255) Previously, it was always unsigned.
- Bug fixes in `CONCAT()` and `WEEKDAY()`.
- Changed a lot of source to get `mysqld` to be compiled with SunPro compiler.
- SQL functions must now have a `'('` immediately after the function name (no intervening space). For example, `'user('` is regarded as beginning a function call, and `'user ('` is regarded as an identifier `user` followed by a `'('`, not as a function call.

D.3.49 Modifications de la version 3.20.0

- The source distribution is done with `configure` and Automake. It will make porting much easier. The `readline` library is included in the distribution.
- Separate client compilation: the client code should be very easy to compile on systems which don't have threads.
- The old Perl interface code is automatically compiled and installed. Automatic compiling of `DBD` will follow when the new `DBD` code is ported.
- Dynamic language support: `mysqld` can now be started with Swedish or English (default) error messages.
- New functions: `INSERT()`, `RTRIM()`, `LTRIM()` and `FORMAT()`.
- `mysqldump` now works correctly for all field types (even `AUTO_INCREMENT`). The format for `SHOW FIELDS FROM nom_table` is changed so the `Type` column contains information suitable for `CREATE TABLE`. In previous releases, some `CREATE TABLE` information had to be patched when recreating tables.
- Some parser bugs from 3.19.5 (`BLOB` and `TIMESTAMP`) are corrected. `TIMESTAMP` now returns different date information depending on its create length.
- Changed parser to allow a database, table or field name to start with a number or `'_'`.
- All old C code from Unireg changed to C++ and cleaned up. This makes the daemon a little smaller and easier to understand.
- A lot of small bug fixes done.
- New INSTALL files (not final version) and some info regarding porting.

D.4 Modifications de la version 3.19.x

D.4.1 Modifications de la version 3.19.5

- Some new functions, some more optimization on joins.
- Should now compile clean on Linux (2.0.x).
- Added functions `DATABASE()`, `USER()`, `POW()`, `LOG10()` (needed for ODBC).
- In a `WHERE` with an `ORDER BY` on fields from only one table, the table is now preferred as first table in a multi−join.
- `HAVING` and `IS NULL` or `IS NOT NULL` now works.
- A group on one column and a sort on a group function (`SUM()`, `AVG()`...) didn't work together. Fixed.
- `mysqldump`: Didn't send password to server.

D.4.2 Modifications de la version 3.19.4

- Fixed horrible locking bug when inserting in one thread and reading in another thread.
- Fixed one−off decimal bug. 1.00 was output as 1.0.
- Added attribute `'Locked'` to process list as info if a query is locked by another query.
- Fixed full magic timestamp. Timestamp length may now be 14, 12, 10, 8, 6, 4 or 2 bytes.
- Sort on some numeric functions could sort incorrectly on last number.
- `IF(arg,syntax_error,syntax_error)` crashed.
- Added functions `CEILING()`, `ROUND()`, `EXP()`, `LOG()` and `SQRT()`.
- Enhanced `BETWEEN` to handle strings.

D.4.3 Modifications de la version 3.19.3

- Fixed `SELECT` with grouping on `BLOB` columns not to return incorrect `BLOB` info. Grouping, sorting and distinct on `BLOB` columns will not yet work as expected (probably it will group/sort by the first 7 characters in the `BLOB`). Grouping on formulas with a fixed string size (use `MID()` on a `BLOB`) should work.
- When doing a full join (no direct keys) on multiple tables with `BLOB` fields, the `BLOB` was garbage on output.
- Fixed `DISTINCT` with calculated columns.

E Erreurs connues et manques de *MySQL*

- Impossible de créer un autre dossier lors de l'utilisation de MIT−pthreads. Etant donné que ceci requiert une modification de MIT−pthreads, nous ne pouvons pas corriger ce problème.
- Les valeurs de type `BLOB` ne peuvent pas être utilisées de manière ``sûre'' dans les clauses `GROUP BY`, `ORDER BY` ou `DISTINCT`. Seuls, les `max_sort_length` premiers octets (par défaut 1024) sont utilisés dans les comparaisons entre `BLOB`. Cela peut être changé avec l'option `−O max_sort_length` de `mysqld`. Une solution pour contourner le problème est d'utiliser une sous chaîne : `SELECT DISTINCT LEFT(blob,2048) FROM nom_table`.
- Les calculs sont fait avec `BIGINT` ou `DOUBLE` (les deux font 64 bits de long). La précision de calcul dépend de la fonction. La régle générale est que les fonctions de bits sont fait en précision `BIGINT`, `IF` et `ELT()` avec une précision `BIGINT` ou `DOUBLE`, et le reste est fait en précision `DOUBLE`. Il vaut mieux éviter d'utiliser les valeurs non signées, supérieures à 63 bits(9223372036854775807) pour tout ce qui n'est pas champs de bits.
- Avant ***MySQL*** 3.23 tous tous les types numériques étaient traités comme des champs à virgule fixées. Cela signifie que vous deviez préciser le nombre de décimales d'un nombre à virgule flottante. Les résultats étaient retournés avec un nombre corrects de décimale.
- Toutes les colonnes, hormis `BLOB` et `TEXT`, sont automatiquement débarassées de leurs espaces de fin de chaînes. Pour les types `CHAR` c'est bon, et peut même être vu comme une fonctionnalité, d'après la norme ANSI SQL92. Le bug est que ***MySQL*** traite les colonnes `VARCHAR` de la même façon.
- ***MySQL*** limite à 255 colonnes de type `ENUM` et `SET` dans une seule table.
- Avant ***MySQL*** 3.23.2, une commande `UPDATE` qui modifiait une clé avec une clause `WHERE` sur la même clé pouvait échouer car la clé était utilisée pour rechercher les lignes, et la même ligne risque d'avoir été trouvée plusieurs fois.
  ```
  UPDATE nom_table SET KEY=KEY+1 WHERE KEY 62; 100;
  ```

  Pour contourner le problème, on pourvait utiliser la requête suivante :

  ```
  mysql62; UPDATE nom_table SET KEY=KEY+1 WHERE KEY+0 62; 100;
  ```

  Cela va fonctionner, car ***MySQL*** ne va pas utiliser d'index dans les expressions qui sont dans la clause `WHERE`.

- `safe_mysqld` redirige tous les messages de `mysqld` vers l'historique `mysqld`. Le problème est que si vous exécutez la commande `mysqladmin refresh` pour fermer et réouvrir l'historique, `stdout` et `stderr` continue de pointer sur l'ancien fichier d'historique. Si vous utilisez `−−log` extensivement, il vaut mieux éditer le fichier `safe_mysqld` et y mettre `` `'hostname'.err' `` au lieu de `` `'hostname'.log' `` pour que vous puissiez facilement récupérer l'espace de l'ancien fichier d'historique, en effacant l'ancient, et exécutant `mysqladmin refresh`.
- Dans la commande `UPDATE`, les colonnes sont modifiées de gauche à droite. Si vous faites référence à une colonne modifiée, vous allez utiliser la valeur modifiée, au lieu d'utiliser la valeur originale. Par exemple :
  ```
  mysql62; UPDATE nom_table SET KEY=KEY+1,KEY=KEY+1
  ```

va mettre dans `KEY` la valeur `2` au lieu de `1`.

Pour les bugs spécifiques à chaque plate forme, reportez vous à la section sur la compilation et le portage.

F Liste de voeux pour les versions futures de MySQL (la TODO)

Tout ce qui est dans cette liste est dans l'ordre de priorité. Si vous voulez pouvoir modifier l'ordre des priorités, prenez une licence ou un contrat de support, et dites nous ce qui vous manque le plus. 3 Support et licences MySQL.

F.1 Fonctionnalités prioritaires

Sous sélections. `select id from t where grp in (select grp from g where u > 100)`

- Lors d'une commande `ALTER TABLE` sur une table qui est sous la forme d'un lien symbolique sur un autre disque, commencer par créer une table temporaire sur le disque actuel.
- `RENAME table as table, table as table [,...]`
- FreeBSD et MIT−pthreads; Est ce que les threads endormis consomme des ressources CPU?
- Autoriser la jointure sur des parties de clés (question d'optimisation).
- Ajouter l'entrée `DECRYPT()`.
- Conserver les définitions de clés étrangères (`FOREIGN`) dans le fichier `` `.frm' ``.
- Curseurs coté serveurs.
- Autoriser les utilisateurs à modifier les options de démarrage.
- Ajouter l'option −−ansi à ***MySQL*** (pour transformer || en CONCAT())
- Ne pas ajouter automatiquement de valeur `DEFAULT` aux colonnes. Au contraire, retourner une erreur lors de l'insertion (`INSERT`) de valeur dans des colonnes sans valeur par défaut.
- Mettre les résultats et les requêtes en cache. Cela sera fait dans un module séparé, qui examinera les requêtes, et retournera les résultats déjà présents dans le cache. Lors de la modification d'une table, cela devra ne supprimer que le nombre minium de requêtes du cache. Cela devrait donner un bon cout d'accélérateur aux machines avec beaucoup de RAM, où les requêtes sont souvent repétées (comme pour les applications WWW). Une première approche serait de ne mettre en cache que les requêtes du type : `SELECT CACHED ....`
- Corriger `` `libmysql.c' `` pour permettre plusieurs commandes `mysql_query()` simultanées, sans lire les résultats, ou bien retourner une alerte lorsque cela se passe.
- Optimiser le type `BIT` pour qu'il ne prenne qu'un 1 bit (actuellement, `BIT` prend un char).
- Comprendre pourquoi MIT−pthreads `ctime()` ne fonctionne pas avec certaines configurations FreeBSD.
- Vérifier que les threads verrouillés ne consomme pas de CPU inutilement.
- Ajouter `ORDER BY` pour les modifications. Cela serait bien pratique avec des fonctions telles que `generate_id(start,step)`.
- Ajouter une option `IMAGE` pour les commandes `LOAD DATA INFILE` afin qu'elle ne modifie pas les champs `TIMESTAMP` et `AUTO_INCREMENT`.
- Faire que `LOAD DATA INFILE` comprenne les syntaxes du type :

```
LOAD DATA INFILE 'Nom_fichier.txt' INTO TABLE nom_table
TEXT_FIELDS (champs_texte1, champs_texte2, champs_texte3)
SET table_field1=concatenate(champs_texte1, champs_texte2), champs_texte3=23
IGNORE text_field3
```

- Autoriser `MIN()`, `MAX()` avec des chaînes (pas dans les fonctions de groupes). Elles seraient alors synonymes de `LEAST()`, `GREATEST()`.
- Démonstration de procédures : analyze
- Exportation automatique de `mysql` vers netscape.
- `LOCK DATABASES`. (avec diverses options).
- `NATURAL JOIN`.
- Modifier le tri pour permettre l'allocation de mémoire en ``hunks" (NDT : paquets? grappes?) pour optimiser l'utilisation de mémoire.
- `DECIMAL` et `NUMERIC` ne peuvent pas comprendre les notations exponentielles : `Field_decimal::store(const char *from,uint len)` doivent être recodées pour corriger ceci.
- Corriger `mysql.cc` pour faire moins de `malloc()` lors des hashages des noms de champs.
- Fonctions: ADD_TO_SET(value,set) et REMOVE_FROM_SET(value,set)
- Ajouter l'utilisation de `t1 JOIN t2 ON ...` et `t1 JOIN t2 USING ...` Actuellement, c'est uniquement possible avec `LEFT JOIN`.
- Ajouter le support complet du type `unsigned long long`.
- Fonction `CASE`.
- Ajouter de nombreuses variables pour `show status`. Notamment : Commandes `INSERT`/`DELETE`/`UPDATE`. Lignes lues et modifiées. SELECT sur 1 table et SELECT avec jointures.Nombre moyen de tables dans un SELECT. Nombre de lecture écriture du buffer de clé (logiques et réél). `ORDER BY`, `GROUP BY`, tables temporaires créées.
- Si `mysql` est interrompu au milieu d'une requête, on devrait pouvoir être capable d'ouvrir une autre connexion, et de tuer l'ancienne requête. Alternativement, une tentative doit être faite pour detecter ceci au niveau du serveur.
- Ajouter une interface pour gérer les informations sur les tables, pour les utiliser comme un système de table. Cela ralentira les traitements,

si vous réclamez les informations sur toutes les tables, mais cela sera beaucoup plus flexible. `SHOW INFO FROM nom_table` devrait être implémenté pour accéder à des informations basiques.
- Permettre à `mysqld` de supporter plusieurs jeu de caractères en même temps.
- Ajouter le support d'UNICODE.
- `NATURAL JOIN`.
- Commande issue d'Oracle : `CONNECT BY PRIOR ...` pour rechercher dans les structures hiérarchiques.
- `RENAME DATABASE`
- `mysqladmin copy database new−database`.
- La liste de processus devrait montrer le nombre de requêtes par thread.
- Option `IGNORE` pour la commande `UPDATE` (Elle effacerait les lignes qui ont une clé en doublon, lors de la modification).
- `select distinct a from foo order by b` ne devrait pas retourner les lignes en doubles, Si il y a différents couples (a,b)
- Changer le format de `DATETIME` pour enregistrer les fractions de seconds.
- Ajouter tous les types manquants de ANSI92 et ODBC 3.0.
- Changer les noms de tables en `NULL` pour les calculs de colonnes.

F.2 Liste nécessaire

- Implementer la fonction: `get_changed_tables(timeout,table1,table2,...)`
- Implementer la fonction: `LAST_UPDATED(nom_table)`
- Modifications atomiques : y compris un langage réutilisable pour les procédures stockées.
- `update items,month set items.price=month.price where items.id=month.id;`
- Changer le mode de lecture des tables pour qu'elles utilisent les tables de mémoires (memmap) à chaque fois que c'est possible. Actuellement, seules les tables compressées utilisent une memmap.
- Ajouter un nouveau droit *'**Show_priv**'* pour la commande `SHOW`.
- Rendre le code automatique des timestamp plus agréable. Ajouter timestamps dans l'historique de mise à jour avec l'option `SET TIMESTAMP=#;`
- Utiliser la lecture/écriture mutex à chaque fois que cela permet de faire gagner du temps.
- Ajouter le support des clés étrangères. On préférera peut être un langage procédural d'abord.
- Des vues simples : d'abord sur une table, puis sur des requêtes complexes).
- Refermer automatiquement une table si une table, une table temporaire, ou un fichier temporaire émet l'erreur 23 (not enough open files : pas assez de fichiers ouverts).
- Lorsqu'on rencontre un champs field=#, affecter toutes les occurences du champs field à #. Actuellement, cela ne fonctionne que pour les cas simples.
- Remplacer toutes les expressions constantes par des expressions calculées, lorsque c'est possible.
- Expression pour optimiser les clés. Actuellement, seuls key = field et key = constant sont optimisées.
- Ajouter quelques fonctions de copie, pour améliorer le code.
- Remplacer `` `sql_yacc.yy' `` par un analyseur inline pour réduire sa taille, et retourner des messages d'erreur plus parlants (5 jours).
- Modifier l'analyseur pour qu'il n'utilise qu'une règle par nombre d'argument différent d'une fonction.
- Utiliser les calculs complets de nom dans la clause d'ordre (Pour ACCESS97)
- `UNION`, `MINUS`, `INTERSECT` et `FULL OUTER JOIN`. (Actuellement seul `LEFT OUTER JOIN` est supporté)
- Permettre `UNIQUE` sur des champs qui peuvent être `NULL`.
- `SQL_OPTION MAX_SELECT_TIME=#` pour mettre une limite de temps à une requête.
- Inscrire l'historique de modification dans une base de donnes.
- Valeur négative pour que `LIMIT` commence à lire les résultats depuis la fin.
- Alarmes pour les fonctions clients client connect/read/write.
- Faire une version de `mysqld` qui ne soit pas multithreaded (3−5 jours).
- Notez le changement de `safe_mysqld`: suivant la norme FSSTND (que Debian essaie de suivre) les fichiers PID doivent être stockés dans `` `/var/run/<nom_du_programme>.pid' `` et les fichiers de logs dans `` `/var/log' ``. Il serait pas mal de pouvoir mettre "DATADIR" dans la première déclaration d'un "pidfile" et "log" : cela permettrait de modifier l'emplacement de ces fichiers d'une seule commande.
- Meilleure gestion des lignes à taille variable, pour éviter la fragmentation.
- `UPDATE SET blob=read_blob_from_file('my_gif') where id=1;`
- Permettre à un client d'enregistrer une requête.
- Ajouter l'utilisation de `zlib()` pour dégzipper les fichiers avec `LOAD DATA INFILE`.
- Finir de corriger le trie et le regroupement de colonne de type `BLOB`.
- Procédures stockées. Cela n'est pas prioritaire pour le moment, car les procédures stockées ne sont pas très standardisées. Un autre problèmes est que les véritables procédures stockées rendent la vite très difficile à l'optimiseur, et dans de nombreux cas, le rende plus lent. D'un autre coté, nous allons écrire un langage atomique et simple de modifications qui pourra être utilisé pour écrire des boucles et autres avec ***MySQL***.
- Utiliser des sémaphores lors du compte des threads. Il faut qu'une librairie de sémaphore soit implémentée pour MIT−pthreads.
- Ne pas assigner de nouvelle valeur à une colonne de type `AUTO_INCREMENT` lorsque la valeur passée est 0. Utiliser plutôt le code `NULL`.
- Ajouter le support complet pour les `JOIN` avec parenthèses.
- Réutiliser les threads pour les systèmes avec de nombreuses connexions.

Le temps est donné en quantité de travail, et non pas en temps réel. Le coeur de métier de TcX's est l'utilisation de ***MySQL*** et non pas son developpement. Mais comme TcX est une société très souple, elle a attribué beaucoup de ressources au developpement de ***MySQL***.

F.3 Liste à long terme

- Transactions avec rollback (Nous utilisons principalement des `SELECT`s, et comme nous n'utilisons pas de transactions, nous pouvons être bien plus rapide que les autres serveurs. Nous allons accepter des opérations atomiques sur les tables multiples. Actuellement, des opérations atomiques peuvent être faites avec `LOCK TABLES`/`UNLOCK TABLES` mais nous rendrons cette manipulation automatique dans le futur.

G Commentaires sur le portage vers d'autres systèmes d'exploitation

A working Posix thread library is needed for the server. On Solaris 2.5 we use SUN PThreads (the native thread support in 2.4 and earlier versions are not good enough) and on Linux we use LinuxThreads by Xavier Leroy, Xavier.Leroy@inria.fr.

The hard part of porting to a new Unix variant without good native thread support is probably to port MIT-pthreads. See `mit-pthreads/README' and Programming POSIX Threads.

The ***MySQL*** distribution includes a patched version of Provenzano's Pthreads from MIT (see MIT Pthreads web page). This can be used for some operating systems that do not have POSIX threads.

It is also possible to use another user level thread package named FSU Pthreads (see FSU Pthreads home page). This implementation is being used for the SCO port.

See the `thr_lock.c' and `thr_alarm.c' programs in the `mysys' directory for some tests/examples of these problems.

Both the server and the client need a working C++ compiler (we use `gcc` and have tried SparcWorks). Another compiler that is known to work is the Irix `cc`.

To compile only the client use `./configure --without-server`.

There is currently no support for only compiling the server. Nor is it likly to be added unless someone has a good reason for it.

If you want/need to change any `Makefile' or the configure script you must get Automake and Autoconf. We have used the `automake-1.2` and `autoconf-2.12` distributions.

All steps needed to remake everything from the most basic files.

```
/bin/rm */.deps/*.P
/bin/rm -f config.cache
aclocal
autoheader
aclocal
automake
autoconf
./configure --with-debug --prefix='your installation directory'

# The makefiles generated above need GNU make 3.75 or newer.
# (called gmake below)
gmake clean all install init-db
```

If you run into problems with a new port, you may have to do some debugging of ***MySQL***! G.1 Debugguer un serveur MySQL.

Note: Before you start debugging `mysqld`, first get the test programs `mysys/thr_alarm` and `mysys/thr_lock` to work. This will ensure that your thread installation has even a remote chance to work!

G.1 Debugguer un serveur MySQL

If you are using some functionality that is very new in ***MySQL***, you can try to run mysqld with the `--skip-new` (which will disable all new, potentially unsafe functionality) or with `--safe-mode` which disables a lot of optimization that may cause problems. 18.1 Que faire si MySQL plante constamment?.

If `mysqld` doesn't want to start, you should check that you don't have any `my.cnf` file that interferes with your setup! You can check your `my.cnf` arguments with `mysqld --print-defaults` and avoid using them by starting with `mysqld --no-defaults ...`.

If you have some very specific problem, you can always try to debug ***MySQL***. To do this you must configure ***MySQL*** with the option `--with-debug`. You can check whether or not ***MySQL*** was compiled with debugging by doing: `mysqld --help`. If the `--debug` flag is listed with the options then you have debugging enabled. `mysqladmin ver` also lists the `mysqld` version as `mysql ... -debug` in this case.

If you are using gcc or egcs, the recommended configure line is:

```
CC=gcc CFLAGS="-O6" CXX=gcc CXXFLAGS="-O6 -felide-constructors -fno-exceptions
-fno-rtti" ./configure --prefix=/usr/local/mysql --with-debug
```

This will avoid problems with the libstdc++ library and with C++ exceptions.

If you can cause the `mysqld` server to crash quickly, you can try to create a trace file of this:

Start the `mysqld` server with a trace log in `` `/tmp/mysql.trace' ``. The log file will get very *BIG*.

```
mysqld --debug --log
```

or you can start it with

```
mysqld --debug=d,info,error,query,general,where:O,/tmp/mysql.trace
```

which only prints information with the most interesting tags.

When you configure ***MySQL*** for debugging you automatically enable a lot of extra safety check functions that monitor the health of `mysqld`. If they find something ``unexpected,'' an entry will be written to `stderr`, which `safe_mysqld` directs to the error log! This also means that if you are having some unexpected problems with ***MySQL*** and are using a source distribution, the first thing you should do is to configure ***MySQL*** for debugging! (The second thing, of course, is to send mail to mysql@lists.mysql.com and ask for help. Please use the `mysqlbug` script for all bug reports or questions regarding the ***MySQL*** version you are using!

On most system you can also start `mysqld` from `gdb` to get more information if `mysqld` crashes.

With some `gdb` versions on Linux you must use `run --one-thread` if you want to be able to debug `mysqld` threads. In this case you can only have one thread active at a time.

If you are using gdb 4.17.x on Linux, you should install a `` `.gdb' `` file, with the following information, in your current directory:

```
set print sevenbit off
handle SIGUSR1 nostop noprint
handle SIGUSR2 nostop noprint
handle SIGWAITING nostop noprint
handle SIGLWP nostop noprint
handle SIGPIPE nostop
handle SIGALRM nostop
handle SIGHUP nostop
handle SIGTERM nostop noprint
```

Here follows an example how to debug mysqld:

```
shell62; gdb /usr/local/libexec/mysqld
gdb62; run
...
back   # Do this when mysqld crashes
info locals
up
info locals
up
...
(until you get some information about local variables)

quit
```

Include the above output in a mail generated with `mysqlbug` and mail this to `mysql@lists.mysql.com`.

If `mysqld` hangs you can try to use some system tools like `strace` or `/usr/proc/bin/pstack` to examine where `mysqld` has hanged.

If `mysqld` starts to eat up CPU or memory or if it ``hangs'', you can use `mysqladmin processlist status` to find out if someone is executing some query that takes a long time. It may be a good idea to run `mysqladmin −i10 processlist status` in some window if you are experiencing performance problems or problems when new clients can't connect.

If `mysqld` dies or hangs, you should start `mysqld` with `−−log`. When `mysqld` dies again, you can check in the log file for the query that killed `mysqld`. Note that before starting `mysqld` with `−−log` you should check all your tables with `isamchk`. 13 Maintenance d'un serveur MySQL.

If you are using a log file, `mysqld −−log`, you should check the 'hostname' log files, that you can find in the database directory, for any requêtesthat could cause a problem. Try the command `EXPLAIN` on all `SELECT` statements that takes a long time to ensure that mysqld are using indexes properly. `EXPLAIN`. You should also test complicated requêtesthat didn't complete within the `mysql` command line tool.

If you find the text `mysqld restarted` in the error log file (normally named `` `hostname.err' ``) you have probably found a query that causes `mysqld` to fail. If this happens you should check all your tables with `isamchk` (13 Maintenance d'un serveur MySQL), and test the requêtesin the ***MySQL*** log files if someone doesn't work. If you find such a query, try first upgrading to the newest ***MySQL*** version. If this doesn't help and you can't find anything in the `mysql` mail archive, you should report the bug to mysql@lists.mysql.com. Links to mail archives are available at the online ***MySQL*** documentation page.

If you get corrupted tables or if `mysqld` always fails after some update commands, you can test if this bug is reproducible by doing the following:

- Stop the mysqld daemon (with `mysqladmin shutdown`)
- Check all tables with `isamchk −s database/*.ISM`. Repair any wrong tables with `isamchk −r database/table.ISM`.
- Start `mysqld` with `−−log−update`
- When you have got a crashed table, stop the `mysqld server`.
- Restore the backup.
- Restart the `mysqld` server ***without*** `−−log−update`
- Re−execute the commands with `mysql < update−log`. The update log is saved in the ***MySQL*** database directory with the name `your−hostname.#`.
- If the tables are now again corrupted, you have found reproducible bug in the `ISAM` code! ftp the tables + the update log to ftp://www.mysql.com/pub/mysql/secret and we will fix this as soon as possible!

The command `mysqladmin debug` will dump some information about locks in use, used memory and query usage to the mysql log file. This may help solve some problems. This command also provides some useful information even if you haven't compiled ***MySQL*** for debugging!

If the problem is that some tables are getting slower and slower you should try to repair the tables with `isamchk` to optimize the table layout. You should also check the slow requêteswith `EXPLAIN`. 13 Maintenance d'un serveur MySQL.

You should also read the OS−specific section in this manual for problems that may be unique to your environment. 4.11 Quelques spécificités liées aux OS

If you are using the Perl `DBI` interface, you can turn on debugging information by using the `trace` méthode or by setting the `DBI_TRACE` environment variable. Perl `DBI` Class.

G.2 Debugguer un client MySQL

To be able to debug a ***MySQL*** client with the integrated debug package, you should configure ***MySQL*** with `−−with−debug`. 4.7.3 Options communes de `configure`

Before running a client, you should set the `MYSQL_DEBUG` environment variable:

```
shell62; MYSQL_DEBUG=d:t:O,/tmp/client.trace
shell62; export MYSQL_DEBUG
```

This causes clients to generate a trace file in `` `/tmp/client.trace' ``.

If you have problems with your own client code, you should attempt to connect to the server and run your query using a client that is known to work. Do this by running `mysql` in debugging mode (assuming you have compiled ***MySQL*** with debugging on):

```
shell62; mysql −−debug=d:t:O,/tmp/client.trace
```

This will provide useful information in case you mail a bug report. 2.3 Comment rapporter des bugs et des problèmes.

If your client crashes at some 'legal' looking code, you should check that your `` `mysql.h' `` include file matches your mysql library file. A very common mistake is to use an old `` `mysql.h' `` file from an old ***MySQL*** installation with new ***MySQL*** library.

G.3 Remarques sur les RTS threads

I have tried to use the RTS thread packages with ***MySQL*** but stumbled on the following problems:

They use old version of a lot of POSIX calls and it is very tedious to make wrappers for all functions. I am inclined to think that it would be easier to change the thread libraries to the newest POSIX specification.

Some wrappers are already written. See `mysys/my_pthread.c' for more info.

At least the following should be changed:

`pthread_get_specific` should use one argument. `sigwait` should take two arguments. A lot of functions (at least `pthread_cond_wait`, `pthread_cond_timedwait`) should return the error code on error. Now they return −1 and set `errno`.

Another problem is that user−level threads use the `ALRM` signal and this aborts a lot of functions (`read`, `write`, `open`...). ***MySQL*** should do a retry on interrupt on all of these but it is not that easy to verify it.

The biggest unsolved problem is the following:

To get thread−level alarms I changed `mysys/thr_alarm.c' to wait between alarms with `pthread_cond_timedwait()`, but this aborts with error `EINTR`. I tried to debug the thread library as to why this happens, but couldn't find any easy solution.

If someone wants to try ***MySQL*** with RTS threads I suggest the following:

- Change functions ***MySQL*** uses from the thread library to POSIX. This shouldn't take that long.
- Compile all libraries with the `−DHAVE_rts_threads`.
- Compile `thr_alarm`.
- If there are some small differences in the implementation, they may be fixed by changing `my_pthread.h' and `my_pthread.c'.
- Run `thr_alarm`. If it runs without any ``warning", ``error" or aborted messages, you are on the right track. Here follows a successful run on Solaris:

```
Main thread: 1
Tread 0 (5) started
Thread: 5  Waiting
process_alarm
Tread 1 (6) started
Thread: 6  Waiting
process_alarm
process_alarm
thread_alarm
Thread: 6  Slept for 1 (1) sec
Thread: 6  Waiting
process_alarm
process_alarm
thread_alarm
Thread: 6  Slept for 2 (2) sec
Thread: 6  Simulation of no alarm needed
Thread: 6  Slept for 0 (3) sec
Thread: 6  Waiting
process_alarm
process_alarm
thread_alarm
Thread: 6  Slept for 4 (4) sec
Thread: 6  Waiting
process_alarm
```

```
thread_alarm
Thread: 5  Slept for 10 (10) sec
Thread: 5  Waiting
process_alarm
process_alarm
thread_alarm
Thread: 6  Slept for 5 (5) sec
Thread: 6  Waiting
process_alarm
process_alarm

...
thread_alarm
Thread: 5  Slept for 0 (1) sec
end
```

G.4 Différences entre les thread packages

MySQL is very dependent on the thread package used. So when choosing a good platform for ***MySQL***, the thread package is very important.

There are at least three types of thread packages:

- User threads in a single process. Thread switching is managed with alarms and the threads library manages all non−thread−safe functions with locks. Read, write and select operations are usually managed with a thread−specific select that switches to another thread if the running threads have to wait for data. If the user thread packages are integrated in the standard libs (FreeBSD and BSDI threads) the thread package requires less overhead than thread packages that have to map all unsafe calls (MIT−pthreads, FSU Pthreads and RTS threads). In some environments (for example, SCO), all system calls are thread−safe so the mapping can be done very easily (FSU Pthreads on SCO). Downside: All mapped calls take a little time and it's quite tricky to be able to handle all situations. There are usually also some system calls that are not handled by the thread package (like MIT−pthreads and sockets). Thread scheduling isn't always optimal.
- User threads in separate processes. Thread switching is done by the kernel and all data are shared between threads. The thread package manages the standard thread calls to allow sharing data between threads. LinuxThreads is using this méthode. Downside: Lots of processes. Thread creating is slow. If one thread dies the rest are usually left hanging and you must kill them all before restarting. Thread switching is somewhat expensive.
- Kernel threads. Thread switching is handled by the thread library or the kernel and is very fast. Everything is done in one process, but on some systems, `ps` may show the different threads. If one thread aborts the whole process aborts. Most system calls are thread−safe and should require very little overhead. Solaris, HP−UX, AIX and OSF1 have kernel threads.

In some systems kernel threads are managed by integrating user level threads in the system libraries. In such cases, the thread switching can only be done by the thread library and the kernel isn't really ``thread aware''.

H Description des expressions régulières sous MySQL

Une expression régulière (regex) est une méthode puissante de rechercher des valeurs complexes.

MySQL utilise l'implémentation de Henry Spencer's. Cela permet d'être compatible avec POSIX 1003.2. ***MySQL*** utilise la version étendue.

Ce chapitre est une approche très simplifiée. Pour avoir tous les détails, allez sur le manuel d'Henry Spencer `regex(7)` qui est inclus dans la distribution. C Contributions à MySQL.

Une expression régulière décrit un ensemble de chaînes. L'expression régulière la plus simple est celle qui ne contient aucun caractère spécial. Par exemple, `bonjour` correspond à `bonjour` et rien d'autre.

Les expressions régulières non triviales utilisent certains caractères spéciaux, ce qui leur permet de décrire plusieurs chaînes. Par exemple, l'expression régulière `bonjour|le|monde` correspond soit à `bonjour`, soit à `le` ou encore à `monde`

Comme exemple plus complexe, la regexp `B[an]*es` correspond à `Bananes`, `Baaaaaes`, `Bes`, `Bans` et n'importe quelle autre chaîne qui commence par `B`, finit par `es`, et contient des `a` et `n` entre.

Une expression régulière peut contenir les caractères spéciaux suivants: caractères/explication:

^

Le début de la chaîne.

```
mysql62; select "fo\nfo" REGEXP "^fo$";           -62; 0
mysql62; select "fofo" REGEXP "^fo";              -62; 1
```

$

La fin de la chaîne.

```
mysql62; select "fo\no" REGEXP "^fo\no$";         -62; 1
mysql62; select "fo\no" REGEXP "^fo$";            -62; 0
```

.

N'importe quel caractère (y compris les nouvelles lignes)

```
mysql62; select "fofo" REGEXP "^f.*";             -62; 1
mysql62; select "fo\nfo" REGEXP "^f.*";           -62; 1
```

*a**

N'importe quel nombre de `a` (0 ou plus).

```
mysql62; select "Ban" REGEXP "^Ba*n";             -62; 1
mysql62; select "Baaan" REGEXP "^Ba*n";           -62; 1
mysql62; select "Bn" REGEXP "^Ba*n";              -62; 1
```

a+

N'importe quel nombre de `a` existants (1 ou plus).

```
mysql62; select "Ban" REGEXP "^Ba+n";             -62; 1
mysql62; select "Bn" REGEXP "^Ba+n";              -62; 0
```

a?

La présence ou l'absence de `a` (0 ou 1).

```
mysql62; select "Bn" REGEXP "^Ba?n";              -62; 1
mysql62; select "Ban" REGEXP "^Ba?n";             -62; 1
mysql62; select "Baan" REGEXP "^Ba?n";            -62; 0
```

de|abc

Alternative : soit de, soit abc.

```
mysql62; select "pi" REGEXP "pi|apa";             −62; 1
mysql62; select "axe" REGEXP "pi|apa";            −62; 0
mysql62; select "apa" REGEXP "pi|apa";            −62; 1
mysql62; select "apa" REGEXP "^(pi|apa)$";        −62; 1
mysql62; select "pi" REGEXP "^(pi|apa)$";         −62; 1
mysql62; select "pix" REGEXP "^(pi|apa)$";        −62; 0
```

*(abc)**

N'importe quel nombre de fois la séquence entre parenthèses.

```
mysql62; select "pi" REGEXP "^(pi)+$";            −62; 1
mysql62; select "pip" REGEXP "^(pi)+$";           −62; 0
mysql62; select "pipi" REGEXP "^(pi)+$";          −62; 1
```

{1}
{2,3}

Une façon plus générale de quantifier un atome

*a**

Peut s'écrire a{0,}.

a+

Peut s'écrire a{1,}.

a?

Peut s'écrire a{0,1}.

Pour être plus précis, un atome suivi par une limite, contenant un entier i et aucune virgule, remplace n'importe quelle séquence de i fois l'atome. Un atome suivi par une limite, contenant un entier i et une virgule, remplace n'importe quelle séquence de i fois ou plus l'atome. Un atome suivi par une limite, contenant un entier i et une virgule et un autre entier j, remplace n'importe quelle séquence de i à j (inclus) fois l'atome. Les deux arguments doivent être 0 >= value <= RE_DUP_MAX (default 255). Si il y a deux arguments, le second doit être supérieur au premier.

[a−dX]
[^a−dX]

Correspond à n'importe quel caractère qui est (ou n'est pas, si ^ est utilisé) soit a, b, c, d or X. Pour inclure le caractère litéral], il doit être placé juste après le crochet ouvrant. Pour inclure le caractère litéral − character, il doit être en premier ou en dernier. De cette façon, [0−9] correspond à tous les chiffres. Tous les autres caractères qui n'ont aucune signification entre [] ne fait que se remplacer lui même.

```
mysql62; select "aXbc" REGEXP "[a−dXYZ]";         −62; 1
mysql62; select "aXbc" REGEXP "^[a−dXYZ]$";       −62; 0
mysql62; select "aXbc" REGEXP "^[a−dXYZ]+$";      −62; 1
mysql62; select "aXbc" REGEXP "^[^a−dXYZ]+$";     −62; 0
mysql62; select "gheis" REGEXP "^[^a−dXYZ]+$";    −62; 1
mysql62; select "gheisa" REGEXP "^[^a−dXYZ]+$";   −62; 0
```

[[.characters.]]

La séquence entre crochés imbriqués sont traités comme une seule élément du crochet père. Ainsi, pour ch l'expressions régulière [[.ch.]]*c correspond bien à chchcc.

[=character−class=]

Une classe d'équivalence, qui remplace tous les éléments équivalent, y compris lui même. Par exemple, si o et (+) sont membre d'une classe d'équivalence, alors [[=o=]], [[=(+)=]], et [o(+)] sont synonymes. Une classe d'équivalence ne peut pas être la fin d'un intervalle.

[:character_class:]

Le nom de la classe de caractères entre [: et :] correspond à la liste de tous les caractères de cette classe. Les noms de classes sont :

alnum	digit	punct
alpha	graph	space
blank	lower	upper
cntrl	print	xdigit

Ils correspondent aux classes de caractères défini à la page ctype(3) du manuel. Localement, on peut disposer d'autres classes. Une classe de caractère ne peut pas être utilisé comme extrémité d'intervalle.

```
mysql62; select "justalnums" REGEXP "[[:alnum:]]+";       −62; 1
mysql62; select "!!" REGEXP "[[:alnum:]]+";               −62; 0
```

[[:<:]]
[[:>:]]

Ces séquences remplacent la chaîne NULL au début et à la fin d'un mot, respectivement. Un mot est défini comme une séquence de caractères de mot qui ne sont ni suivi ni précédé d'autres caractères de mot. Un caracètre de mot est un caractère alpha numérique (comme défine par ctype(3)) et l'underscore (_).

```
      mysql62; select "a word a" REGEXP "[[:60;:]]word[[:62;:]]";      −62; 1
      mysql62; select "a xword a" REGEXP "[[:60;:]]word[[:62;:]]";     −62; 0

mysql62; select "weeknights" REGEXP "^(wee|week)(knights|nights)$"; −62; 1
```

I Qu'est ce que Unireg?

Unireg est le constructeur d'interface tty, qui utilise un niveau de connexion très bas aux fichiers NISAM (utilisés par ***MySQL***) et gr ce à ça, il est très rapide. Il existe depuis 1979 (sous Unix en C depuis 1986).

Unireg est constitué de la façon suivante:

- Une vue de table pour la navigation et les modifications
- Une vue multi tables (avec scrolling).
- Un créateur de table. (Avec de nombreuses balises de colonnes inacessibles avec ***MySQL***) C'est WYSIWYG (pour un tty). Vous concevez un écran, et Unireg vous fait préciser les spécifications de colonnes
- Report generator.
- Nombreux utilitaires (export/import rapides de tables vers/depuis des fichiers textes, analyse de contenu de table...).
- Modifications multi−tables puissantes (que nous utilisons intensivement) comme un langage BASIC avec beaucoup de fonctions.
- Langage dynamiques (uniquement en Suédois et en Finlandais). Si quelqu'un est volontaire pour une traduction en Anglais, il y a quelques fichiers à traduire.
- La possibilité de faire des modifications de manière interactive, ou par batch.
- Définitions de clés comme sous Emacs avec des macros clavier.
- Tout ceci dans un fichier binaire de 800K.
- L'utilitaire `convform`. Converti les fichiers `` `.frm' `` et les fichiers text d'un jeu de caractère à l'autre.
- L'utilitaire `pack_isam` . Compacte uen table NISAM (réduction de 50 à 80%). La table peut être lue par ***MySQL*** comme une table ordinaire. Seul, un enregistrement doit être décompréssé par acces. Ne gère pas les types `BLOB` et `TEXT` ou les modifications (pour le moment).

Nous accédons à nos bases de production avec l'interface Unireg et pour servir les pages web via ***MySQL*** (et dans certains cas extremes, le générateur de rapport Unireg).

Unireg prend environs 3Mo sur le disque, et fonctionne au moins sur les OS suivants : SunOS 4.x, Solaris, Linux, HP−UX, ICL Unix, DNIX, SCO et MS−DOS.

Unireg n'est actuellement disponible qu'en suédois et finlandais.

Le prix de Unireg est de 10,000 couronnes suédoies (environs $1500 US), mais cela inclus le support. Unireg est distributé sous la forme d'un binaire. mais toutes les sources ISAM peuvent être trouvées à ***MySQL***). Généralement, nous compilons le binaire pour chaque client, sur son site.

Tous les nouveaux developpement sont concentrés sur ***MySQL***.

J Licence MySQL pour les sytèmes d'exploitation non−Microsoft

MySQL FREE PUBLIC LICENSE (Version 4, March 5, 1995)

NOTE: This license is not the same as any of the GNU Licenses published by the Free Software Foundation. Its terms are substantially different from those of the GNU Licenses. If you are familiar with the GNU Licenses, please read this license with extra care.

This License applies to the computer program known as "MySQL". The "Program", below, refers to such program, and a "work based on the Program" means either the Program or any derivative work of the Program, as defined in the United States Copyright Act of 1976, such as a translation or a modification. The Program is a copyrighted work whose copyright is held by TcX Datakonsult AB and Monty Program KB and Detron HB.

This License does not apply when running "MySQL" on any Microsoft operating system. Microsoft operating systems include all versions of Microsoft Windows NT and Microsoft Windows.

BY MODIFYING OR DISTRIBUTING THE PROGRAM (OR ANY WORK BASED ON THE PROGRAM), YOU INDICATE YOUR ACCEPTANCE OF THIS LICENSE TO DO SO, AND ALL ITS TERMS AND CONDITIONS FOR COPYING, DISTRIBUTING OR MODIFYING THE PROGRAM OR WORKS BASED ON IT. NOTHING OTHER THAN THIS LICENSE GRANTS YOU PERMISSION TO MODIFY OR DISTRIBUTE THE PROGRAM OR ITS DERIVATIVE WORKS. THESE ACTIONS ARE PROHIBITED BY LAW. IF YOU DO NOT ACCEPT THESE TERMS AND CONDITIONS, DO NOT MODIFY OR DISTRIBUTE THE PROGRAM.

1. Licenses. Licensor hereby grants you the following rights, provided that you comply with all of the restrictions set forth in this License and provided, further, that you distribute an unmodified copy of this License with the Program:
    1. You may copy and distribute literal (i.e., verbatim) copies of the Program's source code as you receive it throughout the world, in any medium.
    2. You may modify the Program, create works based on the Program and distribute copies of such throughout the world, in any medium.
2. Restrictions. This license is subject to the following restrictions:
    1. Distribution of the Program or any work based on the Program by a commercial organization to any third party is prohibited if any payment is made in connection with such distribution, whether directly (as in payment for a copy of the Program) or indirectly (as in payment for some service related to the Program, or payment for some product or service that includes a copy of the Program "without charge"; these are only examples, and not an exhaustive enumeration of prohibited activities). However, the following méthodes of distribution involving payment shall not in and of themselves be a violation of this restriction:
        1. Posting the Program on a public access information storage and retrieval service for which a fee is received for retrieving information (such as an on−line service), provided that the fee is not content−dependent (i.e., the fee would be the same for retrieving the same volume of information consisting of random data).
        2. Distributing the Program on a CD−ROM, provided that the files containing the Program are reproduced entirely and verbatim on such CD−ROM, and provided further that all information on such CD−ROM be redistributable for non−commercial purposes without charge.
    2. Activities other than copying, distribution and modification of the Program are not subject to this License and they are outside its scope. Functional use (running) of the Program is not restricted, and any output produced through the use of the Program is subject to this license only if its contents constitute a work based on the Program (independent of having been made by running the Program).
    3. You must meet all of the following conditions with respect to the distribution of any work based on the Program:
        1. If you have modified the Program, you must cause your work to carry prominent notices stating that you have modified the Program's files and the date of any change;
        2. You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole and at no charge to all third parties under the terms of this License;

    3. If the modified program normally reads commands interactively when run, you must cause it, at each time the modified program commences operation, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty). Such notice must also state that users may redistribute the Program only under the conditions of this License and tell the user how to view the copy of this License included with the Program. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.);
    4. You must accompany any such work based on the Program with the complete corresponding machine−readable source code, delivered on a medium customarily used for software interchange. The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable code. However, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable code;
    5. If you distribute any written or printed material at all with the Program or any work based on the Program, such material must include either a written copy of this License, or a prominent written indication that the Program or the work based on the Program is covered by this License and written instructions for printing and/or displaying the copy of the License on the distribution medium;
    6. You may not impose any further restrictions on the recipient's exercise of the rights granted herein. If distribution of executable or object code is made by offering the equivalent ability to copy from a designated place, then offering equivalent ability to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source code along with the object code.
3. Reservation of Rights. No rights are granted to the Program except as expressly set forth herein. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.
4. Other Restrictions. If the distribution and/or use of the Program is restricted in certain countries for any reason, Licensor may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.
5. Limitations. THE PROGRAM IS PROVIDED TO YOU "AS IS," WITHOUT WARRANTY. THERE IS NO WARRANTY FOR THE PROGRAM, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL LICENSOR, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

K Licence MySQL pour les OS Microsoft

MySQL shareware license for Microsoft operating systems (Version 1, September 4, 1998)

This License applies to the computer program known as "MySQL".

This License applies when running MySQL on any Microsoft operating system. Microsoft operating systems include all versions of Microsoft Windows NT and Microsoft Windows.

YOU SHOULD CAREFULLY READ THE FOLLOWING TERMS AND CONDITIONS BEFORE USING, COPYING OR DISTRIBUTING MySQL. BY USING, COPYING AND DISTRIBUTING MySQL, YOU INDICATE YOUR ACCEPTANCE OF THIS LICENSE TO DO SO, AND ALL ITS TERMS AND CONDITIONS FOR USING, COPYING AND DISTRIBUTING MySQL OR WORKS BASED ON IT. NOTHING OTHER THAN THIS LICENSE GRANTS YOU PERMISSION TO USE, COPY OR DISTRIBUTE MySQL OR ITS DERIVATIVE WORKS. THESE ACTIONS ARE PROHIBITED BY LAW. IF YOU DO NOT ACCEPT THESE TERMS AND CONDITIONS, DO NOT USE, COPY OR DISTRIBUTE MySQL.

1. Evaluation and License Registration. This is an evaluation version of MySQL for Win32. Subject to the terms below, you are hereby licensed to use MySQL for evaluation purposes without charge for a period of 30 days. If you use MySQL after the 30 day evaluation period the registration and purchase of a MySQL license is required. The price for a MySQL license is currently 200 US dollars and email support starts from 200 US dollars/year. Quantity discounts are available. If you pay by credit card, the currency is EURO (The European Unions common currency) so the prices will differ slightly. The easiest way to register or find options about how to pay for MySQL is to use the license form at TcX's secure server at https://www.mysql.com/license.htmy. This can be used also when paying with credit card over the Internet. Other applicable méthodes for paying are SWIFT payments, cheques and credit cards. Payment should be made to:

   ```
   Postgirot Bank AB
   105 06 STOCKHOLM, SWEDEN

   TCX DataKonsult AB
   BOX 6434
   11382 STOCKHOLM, SWEDEN

   SWIFT address: PGSI SESS
   Account number: 96 77 06 - 3
   ```

   Specify: license and/or support and your name and email address. In Europe and Japan, EuroGiro (that should be cheaper) can be used to the same account. If you want to pay by cheque make it payable to "Monty Program KB" and mail it to the address below.

   ```
   TCX DataKonsult AB
   BOX 6434
   11382 STOCKHOLM, SWEDEN
   ```

   For more information about commercial licensing, please contact:

   ```
   David Axmark
   Kungsgatan 65 B
   753 21 UPPSALA
   SWEDEN
   Voice Phone +46-18-10 22 80     GMT 9-21. Swedish and English spoken
   Fax +46-8-729 69 05             Email *much* preferred.
   E-Mail: mysql-licensing@mysql.com
   ```

   For more about the license prices and commercial support, like email support, please refer to the MySQL manual. 3.5 Licences et couts du support MySQL. 3.6 Types de support commercial. The use of MySQL or any work based on MySQL after the 30-day evaluation period is

in violation of international copyright laws.

2. Registered version of MySQL. After you have purchased a MySQL license we will send you a receipt by paper mail. You are allowed to use MySQL or any work based on MySQL after the 30−days evaluation period. The use of MySQL is, however, restricted to one physical computer, but there are no restrictions on concurrent uses of MySQL or the number of MySQL servers run on the computer. We will also email you an address and password for a password−protected WWW page that always has the newest MySQL−Win32 version. Our current policy is that a user with the MySQL license can get free upgrades. The best way to ensure that you get the best possible support is to purchase commercial support!
3. Registration for use in education and university or government−sponsored research. You may obtain a MySQL license for the use in education and university or government−sponsored research for free. In that case, send a detailed application for licensing MySQL for such use to the email address The following information is required in the application:
    - The name of the school or institute.
    - A short description of the school or institute and of the type of education, resarch or other functions it provides.
    - A detailed report of the use of MySQL in the institution.

   In this case you will be provided with a license that entitles you to use MySQL in a specified manner.
4. Distribution. Provided that you verify that you are distributing an evaluation or educational/research version of MySQL you are hereby licensed to make as many literal (i.e., verbatim) copies of the evaluation version of MySQL and documentation as you wish.
5. Restrictions. The client code of MySQL is in the Public Domain or under the GPL (for example the code for readline) license. You are not allowed to modify, recompile, translate or create derivative works based upon any part of the server code of MySQL.
6. Reservation of Rights. No rights are granted to MySQL except as expressly set forth herein. You may not copy or distribute MySQL except as expressly provided under this License. Any attempt otherwise to copy or distribute MySQL is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.
7. Other Restrictions. If the distribution and/or use of MySQL is restricted in certain countries for any reason, the Licensor may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.
8. Limitations. MySQL IS PROVIDED TO YOU "AS IS," WITHOUT WARRANTY. THERE IS NO WARRANTY FOR MySQL, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF MySQL IS WITH YOU. SHOULD MySQL PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL THE LICENSOR, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE MySQL AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE MySQL (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF MySQL TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Index des commandes SQL, types et fonctions

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

Y

Index des concepts

A

B

C

D

E

F

G

H

I

L

M

N

O

P

Q

R

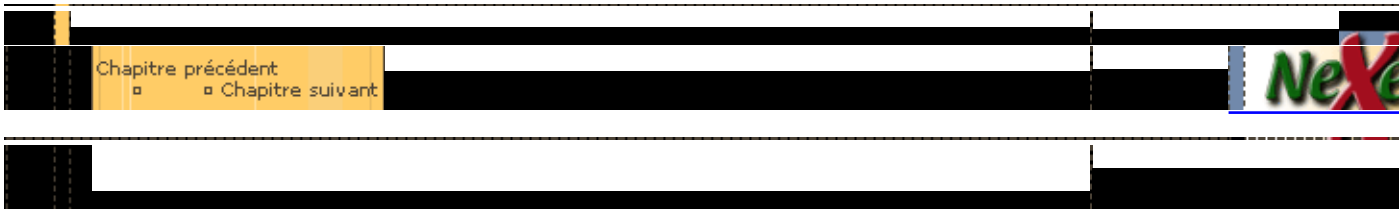
S

T

U

V

W



This document was generated on 16 August 2000 using the texi2html translator version 1.52 (extended by davida@detron.se, hacked by dams@nexen.net for French translation). This document was generated on 16 August 2000 using the texi2html translator version 1.52 (extended by davida@detron.se, hacked by dams@nexen.net for French translation).